# Genetic Algorithms for Numerical Optimization

Zbigniew Michalewicz* Cezary Z. Janikow†

**Abstract**

Genetic algorithms (GAs) are stochastic adaptive algorithms whose search method is based on simulation of natural genetic inheritance and Darwinian strive for survival. They can be used to find approximate solutions to numerical optimization problems in cases where finding the exact optimum is prohibitively expensive, or where no algorithm is known. However, there are some problems that such applications encounter that sometimes delay, if not prohibit, finding the optimal solutions with desired precision. In this paper we describe GAs applications to numerical optimization, present three novel ways to handle such problems, and give some experimental results.

**Keywords:** genetic algorithm, random algorithm, optimization technique, constraint handling, local tuning, convergence.

## 1 Introduction

In the fifties, Von Neumann created a theory of *self-reproducing automata* [Von Neumann (1966)], which put foundations to the field of *genetic algorithms*. In the late fifties Holland continued this idea [Holland (1959)]. In his more recent research [Holland (1975)] he discussed the ability of a simple bit string representation to encode complicated structures and the transformations to improve them. The main result of his work was a demonstration that, with the proper control structure, rapid improvements of bit strings could occur under certain transformations. Even in large and complicated search spaces, given certain conditions on the problem domain, genetic algorithms would tend to converge on solutions that were globally optimal or nearly so.

Genetic algorithms [Davis (1987), [DeJong (1985), Goldberg (1989), Holland (1975)] implement these ideas; they are a class of probabilistic algorithms that start with a population of randomly generated feasible solutions. These solutions "evolve" towards better ones by

*Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA
†Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA

applying genetic operators modeled on the genetic processes occurring in nature. (For a comparison of GA with other optimization methods the reader is referred to [Ackley (1987)]).

GAs have been quite successfully applied to optimization problems like wire routing, scheduling, adaptive control, game playing, cognitive modeling, transportation problem, traveling salesman problem, optimal control problems, *etc.* (see [Goldberg (1989), DeJong (1985), Vignaux & Michalewicz (1989), Vignaux & Michalewicz (1990), Michalewicz et al. (1990), Booker (1982), Michalewicz et al. (1990b), Grefenstette (1985), Grefenstette (1987a), Schaffer (1989)]).

However, as stated in [DeJong (1985)]:

> "...because of this historical focus and emphasis on function optimization applications, it is easy to fall into the trap of perceiving GAs *themselves* as optimization algorithms and then being surprised and/or disappointed when they fail to find an 'obvious' optimum in a particular search space. My suggestion for avoiding this perceptual trap is to think of GAs as a (highly idealized) simulation of a natural process and as such they embody the goals and purposes (if any) of that natural process. I am not sure if anyone is up to the task of defining the goals and purpose of evolutionary systems; however, I think it's fair to say that such systems are *not* generally perceived as functions optimizers".

The purpose of this paper is twofold. Firstly we provide a survey of genetic algorithms, discussing what they are, why they work, and how they work. Secondly, we present various modifications of the classical GAs; these modifications result in a system which can be perceived as a function optimizer.

The paper is organized as follows. In Section 2 we explain the main idea behind genetic algorithms and in Section 3 we describe the major problems that genetic algorithm implementations encounter. Following, the next three Sections 4–6 discuss the major problems and proposed solutions in detail. The discussion is supported by a series of experiments. The last Section gives conclusions and directions for future work.

# 2 Genetic algorithms

In this Section we introduce the genetic algorithms (GAs), present their theoretical foundations, and describe their applicability.

## 2.1 GA: what they are?

GA represent a class of adaptive algorithms whose search methods are based on simulation of natural genetics. They belong to the class of probabilistic algorithms; yet, they are very different from random algorithms as they combine elements of directed and stochastic search. Also, for hard optimization problems, they are superior to *hill–climbing* methods,

since at any time GA provide for both exploitation of the best solutions and exploration of the search space. Because of this, GA are also more robust than existing directed search methods. Another important property of such genetic based search methods is their domain–independence.

In general, a GA performs a multi–directional search by maintaining a population of potential solutions and encourages information formation and exchange between these directions. This population undergoes a simulated evolution: at each generation the relatively "good" solutions reproduce, while the relatively "bad" solutions die. To distinguish between different solutions we need some evaluation function which plays the role of an environment.

**procedure genetic algorithm**
**begin**
  $t = 0$
  initialize $P(t)$
  evaluate $P(t)$
  **while** (**not** termination-condition) **do**
  **begin**
    $t = t + 1$
    select $P(t)$ from $P(t-1)$
    recombine $P(t)$
    evaluate $P(t)$
  **end**
**end**

Figure 1: A simple genetic algorithm.

The structure of a simple genetic algorithm is shown in Figure 1. During iteration $t$, the genetic algorithm maintains a population of potential solutions (called chromosomes following the natural terminology), $P(t) = \{x_1^t, \ldots, x_n^t\}$. Each solution $x_i^t$ is evaluated to give some measure of its "fitness". Then, a new population (iteration $t + 1$) is formed by selecting the more fitted individuals. Some members of this new population undergo reproduction by means of crossover and mutation, to form new solutions.

Crossover combines the features of two parent chromosomes to form two similar offspring by swapping corresponding segments of the parents. For example, if the parents are represented by five-dimensional vectors $\langle a_1, b_1, c_1, d_1, e_1\rangle$ and $\langle a_2, b_2, c_2, d_2, e_2\rangle$ (with each element called a *gene*), then crossing the chromosomes after the second gene would produce the offspring $\langle a_1, b_1, c_2, d_2, e_2\rangle$ and $\langle a_2, b_2, c_1, d_1, e_1\rangle$. The intuition behind the applicability of the crossover operator is information exchange between different potential solutions.

Mutation arbitrarily alters one or more genes of a selected chromosome, by a random change with a probability equal to the mutation rate. The intuition behind the mutation operator is the introduction of some extra variability into the population.

A genetic algorithm for a particular problem must have the following five components:

- a genetic representation for potential solutions to the problem,
- a way to create an initial population of potential solutions,
- an evaluation function that plays the role of the environment, rating solutions in terms of their "fitness",
- genetic operators that alter the composition of children during reproduction,
- values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, *etc.*).

## 2.2 GA: why they work?

The theoretical foundations of genetic algorithms rely on a binary string representation of solutions, and on the notion of a schema (see *e.g.* [Holland (1975)]) — a template allowing exploration of similarities among chromosomes. A schema is built by introducing a new *don't care* symbol ($\star$) into the alphabet of genes — such a schema represents all strings (a hyperplane, or subset of the search space) which match it on all positions other than $\star$. In a population of size $n$ of chromosomes of length $m$, between $2^m$ and $n \cdot 2^m$ different schemata may be represented; at least $n^3$ of them are processed usefully — Holland has called this property an *implicit parallelism*, as it is obtained without of any extra memory/processing requirements.

Two other important notions, associated with the schema, are necessary to derive the theoretical basis:

- **schema order**, $o(H)$, is the number of non *don't care* positions. It defines the speciality of a schema,
- **schema defining length**, $\ell(H)$, is the distance between the first and the last non *don't care* symbols of a chromosome. It defines the compactness of information contained in a schema.

Assuming that the selective probability is proportional to fitness, and independent probabilities $p_c$ and $p_m$ for crossover and mutation, respectively, we can derive the following growth equation (*e.g.* see [Goldberg (1989)]:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H, t)}{\overline{f}(t)} \cdot \left[1 - p_c \frac{\ell(H)}{l-1} - p_m \cdot o(H)\right] \tag{1}$$

where $m(H, t)$ is the number of schema $H$ at time $t$, $f(H, t)$ is the average fitness of schema $H$ at time $t$, and $\overline{f}(t)$ is the average fitness of the population. This equation is also based on the assumption that the fitness function $f$ returns only positive values; when applying GAs to optimization problems where the optimization function may return negative values, some

additional mapping between optimization and fitness functions is required (see [Goldberg (1989)]).

The growth equation (1) shows that selection increases sampling rates of the above–average schemata, and that this change is exponential. The sampling itself does not introduce any new schemata (not represented in the initial $t = 0$ sampling), This is exactly why the crossover operator is introduced — to enable structured, yet random information exchange. Additionally, the mutation operator introduces greater variability into the population. The combined (disruptive) effect of these operators on a schema is not significant if the schema is short and low-order. This result can be stated as:

**Theorem 1 (Schemata Theorem)** *Short, low–order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.*

An immediate result of this theorem is that GA explore the search space by short schemata which, subsequently, are used for information exchange during crossover:

**Hypothesis 1 (Building Block Hypothesis)** *A genetic algorithm seeks near optimal performance through the juxtaposition of short, low–order, high–performance schemata, called the building blocks.*

As stated in [Goldberg (1989)]:

> "Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high preformance schemata, or building blocks".

Although some research has been done to prove this hypothesis (*e.g.* [Bethke (1980)]), for most nontrivial applications we rely mostly on empirical results. During the last 15 years many GA applications were developed which supported the building block hypothesis in many different problem domains. Nevertheless, this hypothesis suggests that the problem of coding for a genetic algorithm is critical for its performance, and that such a coding should satisfy the idea of short building blocks.

Since the binary alphabet offers the maximum number of schemata per bit of information of any coding (see [Goldberg, (1989)]), the bit string representation of solutions has dominated genetic algorithm research. Additionally, such coding provides simplicity of analysis and elegance of available operators. But the "implicit parallelism" result does not depend on the use of bit strings (see [Antonisse & Keller (1987)]) — hence it may be worthwhile to experiment with richer data structures and other "genetic" operators. This may be important in particular in the presence of nontrivial constraints on potential solutions to the problem.

## 2.3 GA: how they work?

Suppose we wish to maximize a function of $k$ variables, $f(x_1, \ldots, x_k) : R^k \rightarrow R$. Suppose further, that each variable $x_i$ can take values from a domain $D_i = [a_i, b_i] \subseteq R$. We wish to optimize the function $f$ with some precision: suppose six decimal places for the variables' values is desirable.

It is clear that to achieve such precision each domain $D_i$ should be cut into $(b_i - a_i) \cdot 10^6$ equal size ranges. Let us denote by $m_i$ the smallest integer such that $(b_i - a_i) \cdot 10^6 \leq 2^{m_i} - 1$. Then, a representation having each variable coded as a binary string of length $m_i$ clearly satisfies the precision requirement. Additionally, the following formula easily interprets each such string:

$$x_i = a_i + decimal(1001...001_2) \cdot \frac{b_i - a_i}{2^{m_i} - 1}$$

where $decimal(string_2)$ represents the decimal value of that binary string.

Now, each chromosome (as a potential solution) is represented by a binary string of length $m = \sum_{i=1}^{k} m_i$; the first $m_1$ bits map into a value from the range $[a_1, b_1]$, the next group of $m_2$ bits map into a value from the range $[a_2, b_2]$, the last group of $m_k$ bits map into a value from the range $[a_k, b_k]$.

To initialize the population, if we do not have any intuition about the distribution of potential optima, we can simply set some *pop-size* number of chromosomes randomly in a bitwise fashion. Otherwise, we can provide some of initial (potential) solutions.

The rest of the algorithm is straightforward: in each generation we evaluate each chromosome (using the function $f$ on the decoded sequences of variables), select new population according to the probability distribution based on fitness values, and recombine the chromosomes in the new population by mutation and crossover operators. After some number of generations, when no further improvement is observed, the best chromosome represents an (possibly the global) optimal solution. In practice, we stop the algorithm after a fixed amount of iterations depending on speed and resource criteria.

## 2.4 GA: an example

Let us assume we wish to find the maximum of the following function:

$$f(x_1, x_2, x_3) = 3.5 \cdot (x_1 - 2.1x_2)^3 - \sqrt{x_1 x_2 + log_2(x_3 + 1)} \cdot sin^2(x_3 + \pi),$$

where $-3.0 \leq x_1 \leq 12.1$, $4.1 \leq x_2 \leq 5.8$, and $0.0 \leq x_3 \leq 50.0$. The required precision is four decimal places for each variable.

The domain of variable $x_1$ has length 15.1; the precision requirement implies that the range $\langle -3.0, 12.1 \rangle$ should be divided into at least $15.1 \cdot 10000$ equal size ranges. It means that 18 bits are required as this part of the chromosome:

$$2^{17} < 151000 \leq 2^{18}$$

The domain of variable $x_2$ has length 1.7; the precision requirement implies that the range $\langle 4.1, 5.8\rangle$ should be divided into at least $1.7 \cdot 10000$ equal size ranges. It means that 15 bits are required as this part of the chromosome:

$$2^{14} < 17000 \leq 2^{15}$$

The domain of variable $x_3$ has length 50.0; the precision requirement implies that the range $\langle 0.0, 50.0\rangle$ should be divided into at least $50.0 \cdot 10000$ equal size ranges. It means that 19 bits are required as this part of the chromosome:

$$2^{18} < 500000 \leq 2^{19}$$

The total length of a chromosome (solution vector) is then $18 + 15 + 19 = 52$ bits; the first 18 bits code $x_1$, bits 19–33 code $x_2$, and bits 34–52 code $x_3$.

Let us consider an example chromosome:

$$0100010010110100001111100101000101010100001000100101$$

The first 18 bits

$$010001001011010000$$

represent $x_1 = -3.0 + decimal(010001001011010000_2) \cdot \frac{12.1-(-3.0)}{2^{18}-1} = -3.0 + 70352 \cdot \frac{15.1}{262143} = -3.0 + 4.052426 = 1.052426$.

The next 15 bits

$$111110010100010$$

represent $x_2 = 4.1 + decimal(111110010100010_2) \cdot \frac{5.8-4.1}{2^{15}-1} = 4.1 + 31906 \cdot \frac{1.7}{32767} = 4.1 + 1.655330 = 5.755330$.

The last 19 bits

$$1010100001000100101$$

represent $x_3 = 0.0 + decimal(1010100001000100101_2) \cdot \frac{50.0-0.0}{2^{19}-1} = 0.0 + 344613 \cdot \frac{50.0}{524288} = 32.864857$.

So the chromosome

$$0100010010110100001111100101000101010100001000100101$$

corresponds to $\langle x_1, x_2, x_3\rangle = \langle 1.052426, 5.755330, 32.864857\rangle$.

The fitness value for this chromosome is

$$f(1.052426, 5.755330, 32.864857) = -4704.580933.$$

To optimize the function $f$ using genetic algorithm, we create a population of such chromosomes (typical population size for such numerical optimization varies from 50 to 100). All 52 bits for each chromosome are initialized randomly. Each chromosome is evaluated and a new population is formed by selecting the more fitted individuals according to their fitness (we normally assume all evaluation are non–negative; there are ways to enforce that). Some chromosomes from the new population would undergo reproduction by means of crossover and mutation, to form new chromosomes (new solutions). For example, two chromosomes:

0100010010110100001111|1001010001010101000010001001 01
0000110111110000010101|0|1000001010100000100001111110 0

may be selected as parents for the crossover; the crossover point is (randomly) selected after the 23rd bit (as marked above); the resulting offspring are

0100010010110100001111|1000001010100000100001111110 0
0000110111110000010101|0|0101000101010100001000100101 01

# 3 Problems with Genetic Algorithms

However, there are some problems that such applications encounter that sometimes delay, if not prohibit, finding the optimal solutions with a desired precision. Such problems originate from many sources as:

- the coding, which moves the operational search space away from the problem space,
- insufficient number of iterations,
- insufficient population size,

*etc.* and manifest themselves as:

- premature convergence of the entire population to a non–global optimum,
- inability to perform fine local tuning,
- inability to operate in the presence of nontrivial constraints.

In this Section we describe these three problems, and in Sections 4–6 we present quite novel ways to handle them.

## 3.1 Premature convergence

The premature convergence is a common problem of genetic algorithms and other optimization algorithms. If convergence occurs too rapidly, then the valuable information developed in part of the population is often lost. Implementations of genetic algorithms are prone to converge prematurely before the optimal solution has been found. As stated in [Booker (1987)]:

> "...While the performance of most implementations is comparable to or better than the performance of many other search techniques, it [GA] still fails to live up to the high expectations engendered by the theory. The problem is that, while the theory points to sampling rates and search behavior in the limit, any implementation uses a finite population or set of sample points. Estimates based on finite samples inevitably have a sampling error and lead to search trajectories much different from those theoretically predicted. This problem is manifested in practice as a premature loss of diversity in the population with the search converging to a sub-optimal solution."

To improve performance of the genetic algorithms, De Jong investigated (see [DeJong (1975)]) five modifications of the basic algorithm. These modifications were called *elitist model*, *expected value model*, *elitist expected value model*, *crowding factor model*, and *generalized crossover model*.

A few years later, Brindle [Brindle (1981)] examined five further modifications: *deterministic sampling*, *remainder stochastic sampling without replacement*, *stochastic sampling without replacement*, *remainder stochastic sampling with replacement*, and *stochastic tournament*. The detailed discussion of all the above modifications is presented in [Goldberg (1989)].

Quite another direction in relaxing this problem borrows ideas from the *simulated annealing* (*e.g.* [Sirag & Weisser (1987)]).

In general, most approaches attempted to improve the convergence of GA presented some modifications to the selection routine.

## 3.2 Fine local tuning

Genetic algorithms display inherent difficulties in performing local search for the numerical applications. As observed in [Grefenstette (1987a)]:

> "Like natural genetic systems, GAs progress by virtue of changing the distribution of high performance substructures in the overall population; individual structures are not the focus of attention. Once the high performance regions of the search space are identified by a GA, it may be useful to invoke a local search routine to optimize the members of the final population."

Local search requires the utilization of schema of higher order and longer defining length than those suggested by the Schema Theorem. Holland suggested (see [Holland (1975)]) that the genetic algorithm should be used as a preprocessor to perform the initial search, before turning the search process over to a system that can employ domain knowledge to guide the local search. Additionally, there are problems where the domains of parameters are unlimited, the number of parameters is quite large, and high precision is required. These requirements imply that the length of the (binary) solution vector is quite significant (for 100 variables with domains in the range $[-500, 500]$, where the precision of six digits after the decimal point is required, the length of the binary solution vector is 3000). For such problems the performance of genetic algorithms is quite poor.

## 3.3 Constraints

The central problem in applications of genetic algorithms is that of constraints; until recently there was not any promising methodology for handling them.

Traditionally, to solve a constrained optimization problem using the genetic algorithm approach we use some penalty functions. However, such approaches suffer from the disadvantage of being tailored to the problem and are not sufficiently general to handle a variety of problems.

In this approach we generate potential solutions without considering the constraints, but incorporating in the evaluation function penalties for suppressing illegal candidates. However, though the evaluation function is usually well defined, there is no accepted methodology for combining it with the penalty [Richardson et al. (1989)]; *e.g.* Davis ([Davis (1987)]) discusses a problem in deciding how large a penalty to impose:

> "If one incorporates a high penalty into the evaluation routine and the domain is one in which production of an individual violating the constraint is likely, one runs the risk of creating a genetic algorithm that spends most of its time evaluating illegal individuals. Further, it can happen that when a legal individual is found, it drives the others out and the population converges on it without finding better individuals, since the likely paths to other legal individuals require the production of illegal individuals as intermediate structures, and the penalties for violating the constraint make it unlikely that such intermediate structure will reproduce. If one imposes moderate penalties, the system may evolve individuals that violate the constraint but are rated better than those that do not because the rest of the evaluation function can be satisfied better by accepting the moderate constraint penalty than by avoiding it".

# 4 Premature convergence

As stated in Section 3, one of the problems encountered by GAs applications is premature convergence of the entire population to a non-global optimum. The premature convergence is closely related to

- the characteristics of the function itself,
- the magnitude and kind of errors introduced by the sampling mechanism.

The problem of premature convergence is primarily related to the existence of local optima, and depends on both function characteristics and sampling of the solution space. For example, assume that $s_v^t \in P(t)$ is close to some local optimum, and $f(s_v^t)$ is much greater than the average evaluation $\overline{f}(t)$. Also, assume that there is no $s_w^t$ close to the global maximum sought; this might be the case for multi–modal functions. In such a case, there is a fast convergence toward that local optimum, with little chance of more global exploration needed to search for other optima. While such a behavior is permissible at the later evolutionary stages, and even desired at the final ones, it is quite disturbing very early. We propose an approach (see [Michalewicz et al. (1990b), Janikow & Michalewicz (1990)]) which diminishes this problem by decreasing the speed of convergence during the early stages of population existence.

The other problem, also reflecting on the convergence, is related to shifts in the average population fitness. Consider two functions: $f_1(s)$, and $f_2(s) = f_1(s) + const$, which have the same relative optima. One would expect that both can be optimized with similar degree of difficulty. However, if $const \gg \overline{f_1}(s)$, then either the function $f_2(s)$ will suffer from much slower convergence than the function $f_1(s)$, or the function $f_1(s)$ will converge possibly to a local optimum. Some of the previous approaches to this problem used rank instead of actual values $f(s_v^k)$ to guide the selective process. Such an approach suffers from some drawbacks. First, it puts the responsibility on the user to decide on the best selection mechanism. Second, it totally ignores information it holds about the absolute evaluations of different chromosomes. Third, it treats all cases uniformly, regardless of the magnitude of the problem.

To deal with such cases we introduce a measure of problematic characteristics of function being optimized, which we later incorporate into the sampling mechanism.

## 4.1 The scaling mechanism

We are mostly interested in some average behavior of the function being optimized. However, we know such behavior only through finite sampling represented in the population. Therefore, we define a measure using statistical random sample variance and mean (we call

such a measure a *span*):

$$s_t = \frac{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(f(x_i^t) - \overline{f}(t)\right)^2}}{\frac{1}{n}\sum_{i=1}^{n} f(x_i^t)}$$

which can be rewritten as:

$$s_t \simeq \sqrt{\frac{n}{n-1}} \cdot \sqrt{\frac{n \cdot \sum_{i=1}^{n} f^2(x_i^t)}{(\sum_{i=1}^{n} f(x_i^t))^2} - 1}$$

This measure is normalized so that it is quite function independent.

Genetic algorithm applications normally require that all chromosomes evaluate to non-negative numbers; such a requirement relates to the sampling mechanism and can be easily enforced. In such a case $max(s_t) = \sqrt{n}$ since $\sum_i^n f_i^2 \leq (\sum_i^n f_i)^2$. However, such a maximal value is obtained only when all but one evaluations are zero — a very unlikely case. Experiments show that most *spans* are less than one. Moreover, we have determined experimentally, by running different functions with differently scaled $s$ and observing the average optimization behavior, that $s^* = 0.1$ gives the best possible trade–off between space exploration and speed of convergence; therefore, we subsequently treat all cases with respect to the relationship between $s_t$ and $s^*$. This particular choice of $s^*$ is rather rough and more extensive experiments are planned to approximate it in a more systematic way. Moreover, it would be interesting to see whether it could be approximated theoretically; however, the mechanism we are about to introduce is rather little affected by some variations of $s^*$.

To preserve efficiency we do not want to recalculate $s_t$ at each iteration. This is actually not necessary as this measure finds the function's characteristics determinable from a random sample. We have determined experimentally that very often the initial population provides a very good approximation. However, if desired, such sampling may be performed for some time before the algorithm actually starts iterating (we call such *span* $s_0$).

For the construction of our scaling mechanism, which we call a *non–uniform power scaling*, we use two dynamic parameters: the *span* $s$ and *population age* $t$ (this parameter is taken to be iteration number of the algorithm). What we seek is a mechanism which produces more random search for small *ages* and increasingly selects only the best chromosomes at late stages of the population life. Moreover, the net effect of such a mechanism should depend on the *span*; higher *span* should cause the whole mechanism to put less emphasis on chromosomes' fitness, somehow randomizing the sampling. The scaling itself uses the *power law scaling*:

$$f_i' = (f_i)^k$$

where $k \sim 0$ forces a random search, while $k > 1$ forces sampling to be allocated to only the most fitted chromosomes. We construct $k$ so that it changes from small to large values over the *population age* (with largest changes very early and very late), and $k$'s magnitude

should be larger and speed of change smaller for problems with lower *span*. The following equation satisfies these goals:

$$k = \left(\frac{s^*}{s_0}\right)^{p_1} \cdot tan^{p_2 \cdot (\frac{s_0}{s^*})^{\alpha}} \left(\frac{t}{T+1} \cdot \frac{\pi}{2}\right)$$

where:

- $p_1$ is a system parameter determining influence of *span* $s$ on magnitude of $k$,
- $p_2$ and $\alpha$ are system parameters determining speed of changes of $k$; $\alpha$ determines how much *span* affects such changes.

## 4.2 The test case

For experimenting with our scaling mechanism we decided to use a single family of functions:

$$f(\vec{x}) = a + \sum_{i=1}^{\#elems} sin(x_i) \cdot sin^{2q}(\frac{i \cdot x_i^2}{\pi}) \quad | \quad x_k \in (0, \pi),\ k = 1, \ldots, \#elems$$

which is $(sin)$–modulating $(\#elems)$–dimensional function of components with nonlinearly increasing frequency. This function, given appropriate parameters $a$ and $m$, could simulate functions of totally different characteristics (as mentioned in Section 3.1):

**A:** $a = 5.0,\ q = 1$. This function, even though nontrivial, exhibits rather nice characteristics: its average *span* $s_0 \simeq 0.1$ matches the most desired $s^*$.

**B** $a = 0.1,\ q = 250$. This function is very difficult to analyze numerically due to its high non–smoothness. Such highly negative undesired characteristics are captured in its $s_0 \simeq 1.0$. Because of such characteristics, any numerical methods, and GA as well, will tend to fall to false optima.

**C:** $a = 1000,\ q = 1$. This function is a constant transformation of the **A** function. Such a shift causes the average *span* to be decreased dramatically to $s_0 \simeq 0.001$.

Note that $f(\vec{x}) > 0$ for $a > 0$. To better visualize such characteristics cross–sections of these three functions are given in Figure 3. For these particular experiments we used $(\#elems) = 10$. Initial $s_0$ for these three functions are given in Figure 2 along with the appropriate behavior of the $k$ exponent.

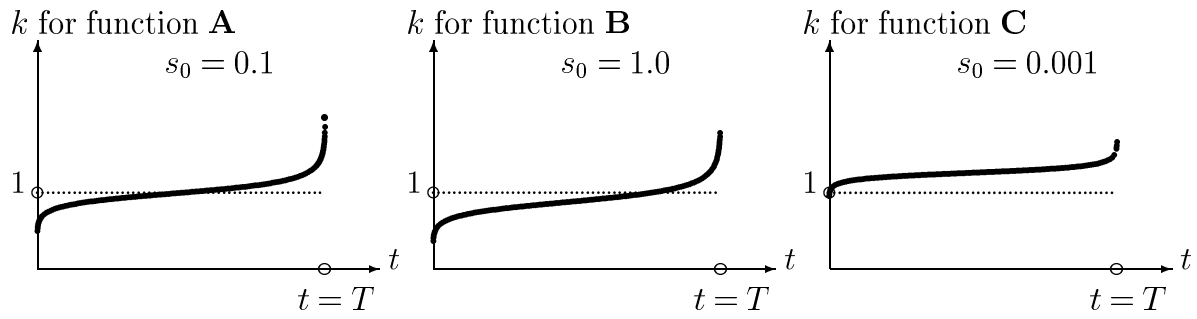


Figure 2: $k$ values for functions **A**, **B**, and **C**: $\alpha = 0.1,\ p_1 = 0.05,\ p2 = 0.1$

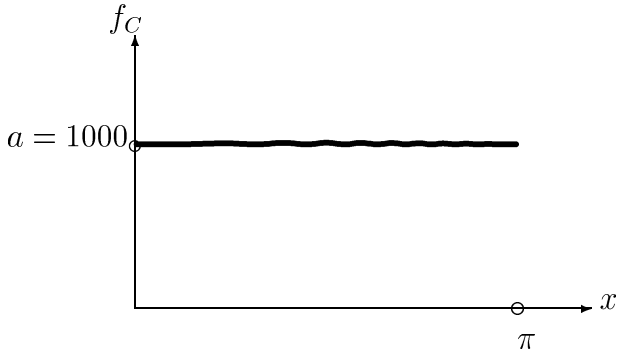


Figure 3: Cross–sections of functions **A**, **B**, and **C** along the $x_1 = \ldots = x_{10}$ plane.

## 4.3 Experiments and results

To judge the quality of our *non–uniform power scaling* we tried to maximize the three functions with both the mechanism on and off. We decided on the following two comparative measures of GA's performance:

- *Accuracy* is defined as the value of the best found chromosome, relative to the value of the global optimum:

$$\frac{f^{best} - a}{f^{global} - a}$$

  We subtract the $a$ parameter so that we could easily compare the results of different functions.

- *Imprecision* is defined as the standard deviation of the optimal vector found. This measure evaluates the closeness of individual components selected in the found optimal chromosome as follow:

$$\sqrt{\frac{\sum_{i=1}^{\#elems}(index_i)^2}{\#elems - 1}}$$

  where $index_i$ is the index of the $i^{th}$ component of the found optimal chromosome (each dimension has its peaks ordered from 0 to the number of dimensions minus one according to the modulated values of its peaks). For example, *imprecision*=0 means the generated chromosome correctly selected the best modulated values along all dimensions; however, it does not have to generate the best function value due to local imperfectness associated with finite iteration time.

We used a traditional binary representation with thirty bits per variable; therefore, a chromosome was a vector of 260 binary bits for $\#elems = 10$. Moreover, the thirty bits gave us precision of $\Delta x = \pi/2^{30} \simeq 3 \cdot 10^{-9}$. This error propagated to function error

$$\Delta f(\vec{x}) = \Delta x \cdot \sum_{i=1}^{\#elems} \frac{\sigma f}{\sigma x_i}$$

which, very pessimistically, can be approximated to

$$\Delta f(\vec{x}) = \Delta x \cdot \sum_{i=1}^{\#elems} (1 + 4 \cdot q \cdot i) = \Delta x \cdot (10 + 220 \cdot q)$$

This, in turn, translates to about $7 \cdot 10^{-7}$ for functions **A** and **C**, and about $1.6 \cdot 10^{-4}$ for function **B**.

During the runs we used one–bit mutations (0.001 rate on bits), single point crossover (0.2 rate on chromosomes), and inversion (0.02 rate on chromosomes). The results are summarized in Table 1; they represent an average of twenty five independent runs, each with population size of forty and iterated 5000 times.

As expected, function **B** turned out to be the most difficult for the GA; its high *span* $s_0 \simeq 1.0$ caused many faulty local optima to be included in the solutions. This is also the case where our scaling mechanism gave best improvements, both in terms is faulty convergence and the absolute magnitude of soltion vectors. The most average function **A** also benefited from the mechanism in terms of both measures; this function has a usual *span*,

| Function | The absolute optimum | $f^{global} - a$ | *Accuracy* | | *Imprecision* | |
|---|---|---|---|---|---|---|
| | | | GA w/o scaling | GA w/ scaling | GA w/o scaling | GA w/ scaling |
| A | 14.70413 | 9.70413 | 98.94 | 99.62 | 0.365 | 0.298 |
| B | 9.75332 | 9.65332 | 92.16 | 94.35 | 1.528 | 1.194 |
| C | 1009.70413 | 9.70413 | 99.42 | 99.51 | 0.258 | 0.257 |

Table 1: Output summary for the three test functions.

but is still highly multi–modal, difficult for any method. The smallest influence was observed on function **C**, the one with a very small *span*; such characteristics prohibited faulty peak selections in a natural way. The increased selective preasure in *older* populations, generated by higher $k$ values, slightly improved *accuracy*, yet preserving high precison.

We are yet to conduct more systematic testing in order to optimize the various parameters used in our scaling mechanism. Nevertheless, these results indicate its usefulness as automatic problem analyzer in cases where the user is not expected to permorf such an analysis.

As to the increased computational complexity of calculating the $k$ parameter, it was rather an insignificant change due to following reasons:

- The span $s_0$ capturing the characteristics of the optimized function was calculated only one at iteration $t = 0$. Moreover, to increase the quality of this measure approximated by the final sampling it was performed without the population size restriction; the number of such samples was set to 200.
- The formula can be partially rewritten to account for constant and incremental part of it.

# 5 Fine local tuning

To improve the fine local tuning capabilities of a genetic algorithm, which is a must for high precision problems, we designed a special mutation operator whose performance is quite different from the traditional one. Recall that a traditional mutation changes one bit of a chromosome at a time; therefore, such a change uses only local knowledge — only the bit undergoing mutation is known. Such a bit, if located in the left portion of a sequence coding a variable, is very significant to the absolute magnitude of the mutation effect on the variable. On the other hand, bits far to the right of such a sequence have quite a smaller influence while mutated. We decided to use such positional global knowledge in the following way: as the population ages, bits located further to the right of each sequence coding one variable

get higher probability of being mutated, while those on the left have such a probability decreasing. In other words, such a mutation causes global search of the search space at the beginning of the iterative process, but only an increasingly local exploitation later on. We call it a *non–uniform mutation.*

For here we actually selected to use a floating point rather than binary representation for the following two reasons:

- Floating point representation is much more natural to implement the *non–uniform mutation* because of equivalency of problem and representation distances.
- Due to fast propagation of errors in the iterative definitions of states (see the test case in Section 5.2), it would be necessary to use quite more than thirty bits per one control state. This, in the presence of forty five coded variables, would create chromosomes of length over 2000 bits — an excessive number for a hope of a reasonable performance.

Such a representation requires appropriately different operators; an interested reader should refer to [Janikow & Michalewicz (1990)].

## 5.1 The non–uniform operator

The *non–uniform mutation* operator was introduced in some our earlier papers (see [Michalewicz & Janikow (1990), Janikow & Michalewicz (1990)]). As mentioned, this is the operator responsible for the fine tuning capabilities in a numerical search space. It is defined as follows: if $s_v^t = \langle v_1, \ldots, v_m \rangle$ is a chromosome and the element $v_k$ was selected for this mutation (domain of $v_k$ is $[l_k, u_k]$), the result is a vector $s_v^{t+1} = \langle v_1, \ldots, v'_k, \ldots, v_m \rangle$, with $k \in \{1, \ldots, n\}$, and

$$v'_k = \begin{cases} v_k + \triangle(t, u_k - v_k) & \text{if a random digit is 0} \\ v_k - \triangle(t, v_k - l_k) & \text{if a random digit is 1} \end{cases}$$

The function $\triangle(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(t, y)$ being close to 0 increases as $t$ increases. This property causes this operator to search the space uniformly initially (when $t$ is small), and very locally at later stages. We have used the following function:

$$\triangle(t, y) = y \cdot \left(1 - r^{(1 - \frac{t}{T})^b}\right),$$

where $r$ is a random number from [0..1], $T$ is the maximal generation number, and $b$ is a system parameter determining the degree of non–uniformity.[1] Figure 4 displays the value of $\Delta$ for two selected times; this picture clearly indicates the behavior of the operator.

[1]For the binary representation we were using $v'_k = mutate(v_k, \nabla(t, m_k))$ where $mutate(v_k, pos)$ means mutate value of variable $k$ on bit $pos$ (0 is the least significant), $m_k$ is the binary length of variable $k$, and

$$\nabla(t, m_k) = \begin{cases} \lfloor \Delta(t, m_k) \rfloor & \text{if a random digit is 0} \\ \lceil \Delta(t, m_k) \rceil & \text{if a random digit is 1} \end{cases}$$

with the $b$ parameter of $\Delta$ adjusted appropriately if similar behavior desired.

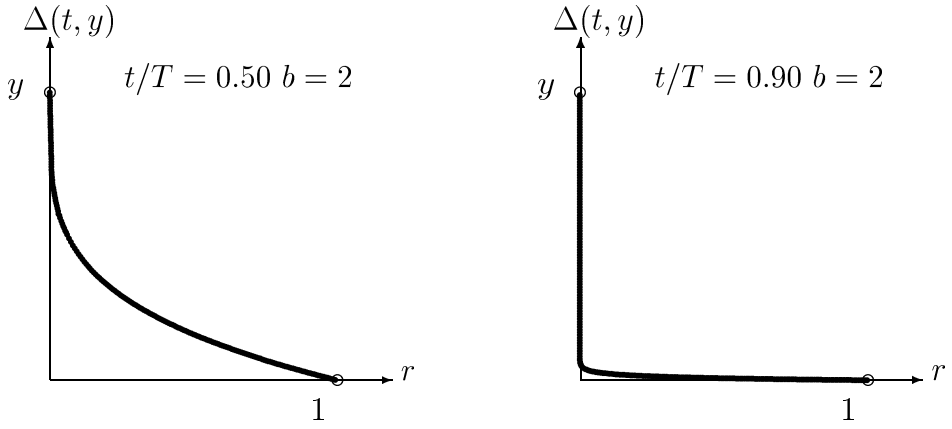


Figure 4: $\triangle(t, y)$ for two selected times.

## 5.2 The test case

To indicate usefulness of the *non–uniform mutation* operator we selected a dynamic control problem without constraints (for further results, especially for a successful comparison with a standard numerical optimization package, see [Janikow & Michalewicz (1990)]).

The problem is a one-dimensional linear-quadratic model:

$$min\left(q \cdot x_N^2 + \sum_{k=0}^{N-1}(s \cdot x_k^2 + r \cdot u_k^2)\right)$$

subject to

$$x_{k+1} = a \cdot x_k + b \cdot u_k,\ k = 0, 1, \ldots, N-1$$

where $x_0$ is a given initial state, $a, b, q, s, r$ are given constants, $x_k \in R$ is a state, and $\vec{u} \in R^N$ is the sought control vector. The optimal value can be analytically expressed as

$$J^* = K_0 x_0^2$$

where $K_k$ is the solution of the Riccati equation

$$K_k = s + ra^2 K_{k+1}/(r + b^2 K_{k+1}) \text{ and } K_N = q$$

## 5.3 Experiments and results

The exact representation is as follow: for a problem of $m$ variables, each chromosome, representing a permissible solution, is represented as a vector of $m$ floating point numbers $s_v^t = \langle v_1, \ldots, v_n \rangle$ (when the generation number $t$ and the chromosome number $i$ are not

| Case | $N$ | $x_0$ | $s$ | $r$ | $q$ | $a$ | $b$ |
|---|---|---|---|---|---|---|---|
| I | 45 | 100 | 1 | 1 | 1 | 1 | 1 |
| II | 45 | 100 | 10 | 1 | 1 | 1 | 1 |
| III | 45 | 100 | 1000 | 1 | 1 | 1 | 1 |
| IV | 45 | 100 | 1 | 10 | 1 | 1 | 1 |
| V | 45 | 100 | 1 | 1000 | 1 | 1 | 1 |
| VI | 45 | 100 | 1 | 1 | 0 | 1 | 1 |
| VII | 45 | 100 | 1 | 1 | 1000 | 1 | 1 |
| VIII | 45 | 100 | 1 | 1 | 1 | 0.01 | 1 |
| IX | 45 | 100 | 1 | 1 | 1 | 1 | 0.01 |
| X | 45 | 100 | 1 | 1 | 1 | 1 | 100 |

Table 2: The ten test cases for the dynamic control problem.

| | Generations | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Case | 1 | 100 | 1,000 | 10,000 | 20,000 | 30,000 | 40,000 | Factor |
| I | 17807.4 | 3.27985 | 1.74689 | 1.61866 | 1.61825 | 1.61804 | 1.61803 | $10^4$ |
| II | 13670.4 | 5.33177 | 1.45968 | 1.11349 | 1.09205 | 1.09165 | 1.09163 | $10^5$ |
| III | 17023.8 | 2.87485 | 1.07974 | 1.00968 | 1.00126 | 1.00104 | 1.00103 | $10^7$ |
| IV | 15077.3 | 8.64310 | 3.75530 | 3.71846 | 3.70812 | 3.70165 | 3.70160 | $10^4$ |
| V | 5956.43 | 12.2559 | 2.89769 | 2.87727 | 2.87646 | 2.87570 | 2.87569 | $10^5$ |
| VI | 16657.7 | 5.07047 | 2.05314 | 1.61869 | 1.61830 | 1.61806 | 1.61806 | $10^4$ |
| VII | 2680666 | 19.2684 | 7.02566 | 1.63464 | 1.62412 | 1.61888 | 1.61882 | $10^4$ |
| VIII | 116.982 | 67.1758 | 1.92764 | 1.00009 | 1.00005 | 1.00005 | 1.00005 | $10^4$ |
| IX | 7.18263 | 4.42849 | 4.37093 | 4.31504 | 4.31024 | 4.31004 | 4.31004 | $10^5$ |
| X | 9870352 | 138132 | 16096.0 | 1.38244 | 1.00041 | 1.00010 | 1.00010 | $10^4$ |

Table 3: Results of the modified genetic algorithm on the dynamic control problem.

important, we write simply $s$). The precision of such a representation is fixed for a given machine, and based on the precision of the floating point (or double, if needed) type.

We experimented with ten different cases as defined in Table 2. For each, we repeated three separate runs of 40,000 generations and reported the best (with respect to final value) in Table 3. The same Table also gives some intermediate results after selected number of generations; for example, the values in column "10,000" indicate the partial results after 10,000 generations, while running 40,000. It is important to note that such values are somehow worse than those obtained while running only 10,000 generation, due to the nature of the *non–uniform* operator. For all cases the population size was fixed at 100. The domain all variables was set as $[-200, 200]$.

It is interesting to compare these results with the exact solutions as well as those obtained from another GA; exactly the same one but without the *non–uniform mutation* on. Table 4 summarizes the results; columns labeled $D$ indicate the relative errors in percents.

| Case | Exact solution value | GA w/ non–uniform mutation value | $D$ | GA w/o non–uniform mutation value | $D$ |
|---|---|---|---|---|---|
| I | 16180.3399 | 16180.3939 | 0.000% | 16234.3233 | 0.334% |
| II | 109160.7978 | 109163.0278 | 0.000% | 113807.2444 | 4.257% |
| III | 10009990.0200 | 10010391.3989 | 0.004% | 10128951.4515 | 1.188% |
| IV | 37015.6212 | 37016.0806 | 0.001% | 37035.5652 | 0.054% |
| V | 287569.3725 | 287569.7389 | 0.000% | 298214.4587 | 3.702% |
| VI | 16180.3399 | 16180.6166 | 0.002% | 16238.2135 | 0.358% |
| VII | 16180.3399 | 16188.2394 | 0.048% | 17278.8502 | 6.786% |
| VIII | 10000.5000 | 10000.5000 | 0.000% | 10000.5000 | 0.000% |
| IX | 431004.0987 | 431004.4092 | 0.000% | 431610.9771 | 0.141% |
| X | 10000.9999 | 10001.0045 | 0.000% | 10439.2695 | 4.380% |

Table 4: Comparison of solutions for the linear-quadratic dynamic control problem.

The genetic algorithm using the *non–uniform mutation* clearly outperforms the other one with respect to the accuracy of the found optimal solution; while the enhanced GA rarely errored by more than few thousands of one percent, the other one hardly ever beat one percent. Moreover, it also converged much faster to that solution (see Table 5 for the case I). In other words it has an additional advantage in time constrained situations.

As an illustration of the *non–uniform mutation*'s effect on the evolutionary process check Figure 5; the new mutation causes quite an increase in the number of improvements observed in the population at the end of the population's life. Moreover, fewer number of such improvements prior to that time, together with an actually faster convergence, clearly indicates a better overall search.

| GA | Generations 1 | 10 | 100 | 1,000 | 10,000 | 40,000 | CPU time |
|---|---|---|---|---|---|---|---|
| w/ *non–uniform mutation* | 178074000 | 293564 | 32798 | 17468 | 16186 | 16180 | 719.4sec[3] |
| w/o *non–uniform mutation* | 17243100 | 415623 | 59872 | 17931 | 17445 | 16234 | 719.1sec[3] |

Table 5: Actual improvements on case I.

In other publications (see [Michalewicz et al. (1990b), Janikow & Michalewicz (1990)]) we present more experiments with other dynamic control problems: these prove superiority of the genetic approaches over commercial optimization packages as well.

[3]Times are similar since the GA w/o the special mutation performed other operations at a higher rate as to achieve same rate of breeding. For a comparison, a GA with a binary representation of thirty bits per variable used 12622sec CPU for the same run (all runs reported from a DEC3100 station).

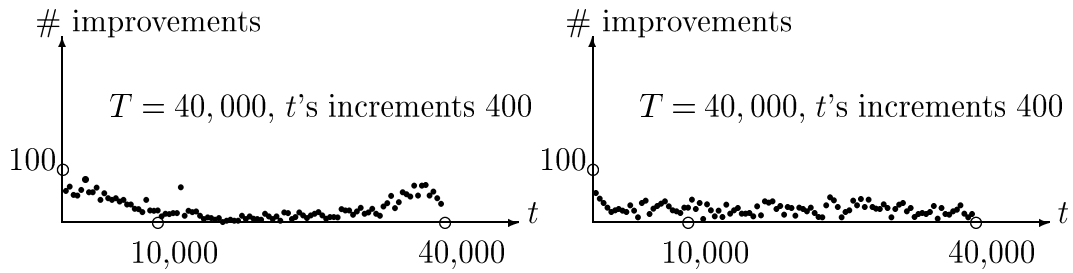


Figure 5: # improvements on case I

# 6 Constraints handling

The proposed methodology for handling linear constraints by genetic algorithms (see [Michalewicz & Janikow (1990)]) combines some previous ideas, but in totally new way and context. The main idea lies in a careful elimination of the equation constraints, and designing *dynamic* operators preserving the inequality constraints. Both types of constraints can be processed very efficiently if they contain only linear equations; so constraints problems include many of the interesting optimization cases.

Linear constraints are of two types: equalities and inequalities (the latter include all variables' domains). We first eliminate all the equalities reducing the number of variables and modifying the inequalities; reducing the set of variables both decreases the length of the representation and reduces the search space. Left with only linear inequalities, we deal with convex search space which, in the presence of dynamic and closed operators, can be searched without explicitly considering the constraints. The problem then becomes one of designing such closed operators. We achieve this by defining them as being context-dependent, that is dynamically adjusting to the current context.

The set of equalities can be eliminated (on one by one basis) as follows:

- transform an equality so that one of its variables is expressed in terms of the others,
- substitute all occurrences of this variable by such an expression, in all remaining equalities and all inequalities.

A detailed description of this approach, along with theoretical foundations, can be found in [Michalewicz & Janikow (1990)]. Here, we provide only an example.

Let us assume we wish to minimize a function of six variables:

$$f(x_1, x_2, x_3, x_4, x_5, x_6)$$

subject to the following constraints:

$$x_1 + x_2 + x_3 = 5$$
$$x_4 + x_5 + x_6 = 10$$
$$x_1 + x_4 = 3$$
$$x_2 + x_5 = 4$$
$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0,\ x_5 \geq 0,\ x_6 \geq 0.$$

We can take an advantage of the presence of four independent equations and express four variables as functions of the remaining two:

$$x_3 = 5 - x_1 - x_2$$
$$x_4 = 3 - x_1$$
$$x_5 = 4 - x_2$$
$$x_6 = 3 + x_1 + x_2$$

Thus, we have reduced the original problem to the optimization problem of two variables $x_1$ and $x_2$:

$$g(x_1, x_2) = f(x_1, x_2, (5 - x_1 - x_2), (3 - x_1), (4 - x_2), (3 + x_1 + x_2)).$$

subject to the following constraints (inequalities only):

$$x_1 \geq 0,\ x_2 \geq 0$$
$$5 - x_1 - x_2 \geq 0$$
$$3 - x_1 \geq 0$$
$$4 - x_2 \geq 0$$
$$3 + x_1 + x_2 \geq 0$$

These inequalities can be further reduced to:

$$0 \leq x_1 \leq 3$$
$$0 \leq x_2 \leq 4$$
$$x_1 + x_2 \leq 5$$

Now, given a chromosome (a point within the constrained solution space), any operator must produce a new feasible solution (this is what we mean by *closedness* of operators). This can be achieved by working within the current context; *e.g.* if $\vec{x} = \langle 1.8, 2.3 \rangle$ is to be mutated on $x_1$, then the new value for this variable must be taken from the range $[0, 5 - x_2] = [0, 2.7]$. This signifies another fact: it is again much easier to define such operators on a floating point representation, as it is not enough to deal locally with a bit at a time.

## 6.1 The dynamic operators

The genetic operators are dynamic, *i.e.* a value of a vector component depends on the remaining values of the vector.

The value of the $i$-th component of a feasible solution $\vec{s} = \langle v_1, \ldots, v_m \rangle$ is always in some (dynamic) range $[l, u]$; the bounds $l$ and $u$ depend on the other vector's values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_m$, and the set of inequalities. We say that the $i$-th component ($i$-th gene) of the vector $\vec{s}$ is *movable* if $l < u$.

Before we describe the operators, we present two important characteristics of convex spaces (due to the linearity of the constraints, the solution space is always a convex space $\mathcal{S}$), which play an essential role in the definition of these operators:

1. For any two points $s_1$ and $s_2$ in the solution space $\mathcal{S}$, the linear combination $a \cdot s_1 + (1 - a) \cdot s_2$, where $a \in [0, 1]$, is a point in $\mathcal{S}$.

2. For every point $s_0 \in \mathcal{S}$ and any line $p$ such that $s_0 \in p$, $p$ intersects the boundaries of $\mathcal{S}$ at precisely two points, say $l_p^{s_0}$ and $u_p^{s_0}$.

Since we are only interested in lines parallel to each axis, to simplify the notation we denote by $l_{(i)}^s$ and $u_{(i)}^s$ the $i$-th components of the vectors $l_p^s$ and $u_p^s$, respectively, where the line $p$ is parallel to the axis $i$. We assume further that $l_{(i)}^s \leq u_{(i)}^s$.

Because of intuitional similarities, we cluster the operators into the standard two classes: mutation and crossover. The proposed crossover and mutation operators use the two properties to ensure that the offspring of a point in the convex solution space $\mathcal{S}$ belongs to $\mathcal{S}$. For a detailed discussion on these topics, the reader is referred to [Michalewicz & Janikow (1990)].

**Mutation group:** Mutations are quite different from the traditional one with respect to both the actual mutation (a gene, being a floating point number, is mutated in a dynamic range) and to the selection of an applicable gene. A traditional mutation is performed on static domains for all genes. In such a case the order of possible mutations on a chromosome does not influence the outcome. This is not true any more with the dynamic domains. To solve the problem we proceed as follows: we randomly select $p_{um} \cdot pop\_size$ chromosomes for uniform mutation, $p_{bm} \cdot pop\_size$ chromosomes for boundary mutation , and $p_{nm} \cdot pop\_size$ chromosomes for non-uniform mutation (all with possible repetitions), where $p_{um}$, $p_{bm}$, and $p_{nm}$ are probabilities of the three mutations defined below. Then, we perform these mutations in a random fashion on the selected chromosome.

- **uniform mutation** for this mutation we select a random gene $k$ (from the set of movable genes of the given chromosome $s$ determined by its current context). If $s_v^t = \langle v_1, \ldots, v_m \rangle$ is a chromosome and the $k$-th component is the selected gene, the result is a vector $s_v^{t+1} = \langle v_1, \ldots, v_k', \ldots, v_m \rangle$, where $v_k'$ is a random value (uniform probability distribution) from the range $[l_{(k)}^{s_v^t}, u_{(k)}^{s_v^t}]$. The dynamic values $l_{(k)}^{s_v^t}$ and $u_{(k)}^{s_v^t}$ are easily calculated from the set of constraints (inequalities).

- **boundary mutation** is a variation of the uniform mutation with $v'_k$ being either $l^{s_v^t}_{(k)}$ or $u^{s_v^t}_{(k)}$, each with equal probability.
- **(dynamic) non-uniform mutation** is a version of the operator defined in Section 5.1; the only change relates to the dynamic domains:

$$v'_k = \begin{cases} v_k + \triangle(t, u^{s_v^t}_{(k)} - v_k) & \text{if a random digit is 0} \\ v_k - \triangle(t, v_k - l^{s_v^t}_{(k)}) & \text{if a random digit is 1} \end{cases}$$

**Crossover group:** Chromosomes are randomly selected in pairs for application of the crossover operators according to appropriate probabilities. The operators defined here are verions of the ones outlined in [Janikow & Michalewicz (1990)] for floating point crossover, but adjusted to our dynamic domains.

- **simple crossover** is defined as follows: if $s_v^t = \langle v_1, \ldots, v_m \rangle$ and $s_w^t = \langle w_1, \ldots, w_m \rangle$ are crossed after the $k$-th position, the resulting offspring are: $s_v^{t+1} = \langle v_1, \ldots, v_k, w_{k+1}, \ldots, w_m \rangle$ and $s_w^{t+1} = \langle w_1, \ldots, w_k, v_{k+1}, \ldots, v_m \rangle$. Note that the only permissible split points are between individual floating points (using float representation it is impossible to split anywhere else).

  However, such operator may produce offspring outside of the convex solution space $\mathcal{S}$. To avoid this problem, we use the property of the convex spaces saying, that there exist $a \in [0, 1]$ such that

$$s_v^{t+1} = \langle v_1, \ldots, v_k, w_{k+1} \cdot a + v_{k+1} \cdot (1 - a), \ldots, w_m \cdot a + v_m \cdot (1 - a) \rangle \in \mathcal{S}$$

  and

$$s_w^{t+1} = \langle w_1, \ldots, w_k, v_{k+1} \cdot a + w_{k+1} \cdot (1 - a), \ldots, v_m \cdot a + w_m \cdot (1 - a) \rangle \in \mathcal{S}$$

  The only question to be answered yet is how to find the largest $a$ to obtain the greatest possible information exchange: due to the real interval, we cannot perform an extensive search. We implemented a binary search (to some depth only for efficiency). Then, $a$ takes the largest appropriate value found, or 0 if no value satisfied the constraints. The necessity for such actions is small in general and decreases rapidly over the life of the population. Note, that the value of $a$ is determined separately for each single arithmetical crossover and each gene.

- **single arithmetical crossover** is defined as follows: if $s_v^t = \langle v_1, \ldots, v_m \rangle$ and $s_w^t = \langle w_1, \ldots, w_m \rangle$ are to be crossed, the resulting offspring are $s_v^{t+1} = \langle v_1, \ldots, v'_k, \ldots, v_m \rangle$ and $s_w^{t+1} = \langle w_1, \ldots, w'_k, \ldots, w_m \rangle$, where $k \in [1, m]$, $v'_k = a \cdot w_k + (1 - a) \cdot v_k$, and $w'_k = a \cdot v_k + (1 - a) \cdot w_k$. Here, $a$ is a dynamic parameter calculated in the given

context (vectors $s_v,\ s_w$) so that the operator is closed (points $x_v^{t+1}$ and $x_w^{t+1}$ are in the convex constrained space $\mathcal{S}$). Actually, $a$ is a random choice from the following range:

$$a \in \begin{cases} [max(\frac{l_{(k)}^{s_w}-w_k}{v_k-w_k}, \frac{u_{(k)}^{s_v}-v_k}{w_k-v_k}), min(\frac{l_{(k)}^{s_v}-v_k}{w_k-v_k}, \frac{u_{(k)}^{s_w}-w_k}{v_k-w_k})] & \text{if } v_k > w_k \\ [0,0] & \text{if } v_k = w_k \\ [max(\frac{l_{(k)}^{s_v}-v_k}{w_k-v_k}, \frac{u_{(k)}^{s_w}-w_k}{v_k-w_k}), min(\frac{l_{(k)}^{s_w}-w_k}{v_k-w_k}, \frac{u_{(k)}^{s_v}-v_k}{w_k-v_k})] & \text{if } v_k < w_k \end{cases}$$

To increase the applicability of this operator (to ensure that $a$ will be non–zero, which actually always nullifies the results of the operator) it is wise to select the applicable gene as a random choice from the intersection of movable genes of both chromosomes. Note again, that the value of $a$ is determined separately for each single arithmetical crossover and each gene.

- **whole arithmetical crossover** is defined as a linear combination of two vectors: if $s_v^t$ and $s_w^t$ are to be crossed, the resulting offspring are $s_v^{t+1} = a \cdot s_w^t + (1-a) \cdot s_v^t$ and $s_w^{t+1} = a \cdot s_v^t + (1-a) \cdot s_w^t$ This operator uses a simpler static system parameter $a \in [0..1]$, as it always guarantees closedness (according to characteristic (1) of this Section).

## 6.2 The test case

For experiments we selected the following problem of forty nine variables: minimize

$$f(\vec{x}) = \sum_{i=1}^{49} g(x_i)\ ,\ \text{where } g(x_i) = \begin{cases} 0,\ if\ \ 0 \le x_i \le 2 \\ c_i,\ if\ \ 2 < x_i \le 4 \\ 2c_i,\ if\ \ 4 < x_i \le 6 \\ 3c_i,\ if\ \ 6 < x_i \le 8 \\ 4c_i,\ if\ \ 8 < x_i \le 10 \\ 5c_i,\ if\ \ 10 < x_i \end{cases}$$

with parameters $c_i$ as given in Table 6 and subject to the following equality constraints:

$$\begin{aligned} &x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = 27 \\ &x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} = 28 \\ &x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} = 25 \\ &x_{22} + x_{23} + x_{24} + x_{25} + x_{26} + x_{27} + x_{28} = 20 \\ &x_{29} + x_{30} + x_{31} + x_{32} + x_{33} + x_{34} + x_{35} = 20 \\ &x_{36} + x_{37} + x_{38} + x_{39} + x_{40} + x_{41} + x_{42} = 20 \\ &x_{43} + x_{44} + x_{45} + x_{46} + x_{47} + x_{48} + x_{49} = 20 \end{aligned}$$

$$\begin{aligned} &x_1 + x_8 + x_{15} + x_{22} + x_{29} + x_{36} + x_{43} = 20 \\ &x_2 + x_9 + x_{16} + x_{23} + x_{30} + x_{37} + x_{44} = 20 \\ &x_3 + x_{10} + x_{17} + x_{24} + x_{31} + x_{38} + x_{45} = 20 \end{aligned}$$

$$x_4 + x_{11} + x_{18} + x_{25} + x_{32} + x_{39} + x_{46} = 23$$
$$x_5 + x_{12} + x_{19} + x_{26} + x_{33} + x_{40} + x_{47} = 26$$
$$x_6 + x_{13} + x_{20} + x_{27} + x_{34} + x_{41} + x_{48} = 25$$
$$x_7 + x_{14} + x_{21} + x_{28} + x_{35} + x_{42} + x_{49} = 26$$

| | | | | | | |
|---|---|---|---|---|---|---|
| $c_1 = 0$ | $c_2 = 21$ | $c_3 = 50$ | $c_4 = 62$ | $c_5 = 93$ | $c_6 = 77$ | $c_7 = 1000$ |
| $c_8 = 21$ | $c_9 = 0$ | $c_{10} = 17$ | $c_{11} = 54$ | $c_{12} = 67$ | $c_{13} = 1000$ | $c_{14} = 48$ |
| $c_{15} = 50$ | $c_{16} = 17$ | $c_{17} = 0$ | $c_{18} = 60$ | $c_{19} = 98$ | $c_{20} = 67$ | $c_{21} = 25$ |
| $c_{22} = 62$ | $c_{23} = 54$ | $c_{24} = 60$ | $c_{25} = 0$ | $c_{26} = 27$ | $c_{27} = 1000$ | $c_{28} = 38$ |
| $c_{29} = 93$ | $c_{30} = 67$ | $c_{31} = 98$ | $c_{32} = 27$ | $c_{33} = 0$ | $c_{34} = 47$ | $c_{35} = 42$ |
| $c_{36} = 77$ | $c_{37} = 1000$ | $c_{38} = 67$ | $c_{39} = 1000$ | $c_{40} = 47$ | $c_{41} = 0$ | $c_{42} = 35$ |
| $c_{43} = 1000$ | $c_{44} = 48$ | $c_{45} = 25$ | $c_{46} = 38$ | $c_{47} = 42$ | $c_{48} = 35$ | $c_{49} = 0$ |

Table 6: Parameters $c_i$.

Given that $\forall_{x_i} x_i \geq 0$, we also have additional implicit domain inequalities, given in Table 7.

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_1 \leq 20$ | $x_2 \leq 20$ | $x_3 \leq 20$ | $x_4 \leq 23$ | $x_5 \leq 26$ | $x_6 \leq 25$ | $x_7 \leq 26$ |
| $x_8 \leq 20$ | $x_9 \leq 20$ | $x_{10} \leq 20$ | $x_{11} \leq 23$ | $x_{12} \leq 26$ | $x_{13} \leq 25$ | $x_{14} \leq 26$ |
| $x_{15} \leq 20$ | $x_{16} \leq 20$ | $x_{17} \leq 20$ | $x_{18} \leq 23$ | $x_{19} \leq 25$ | $x_{20} \leq 25$ | $x_{21} \leq 25$ |
| $x_{22} \leq 20$ | $x_{23} \leq 20$ | $x_{24} \leq 20$ | $x_{25} \leq 20$ | $x_{26} \leq 20$ | $x_{27} \leq 20$ | $x_{28} \leq 20$ |
| $x_{29} \leq 20$ | $x_{30} \leq 20$ | $x_{31} \leq 20$ | $x_{32} \leq 20$ | $x_{33} \leq 20$ | $x_{34} \leq 20$ | $x_{35} \leq 20$ |
| $x_{36} \leq 20$ | $x_{37} \leq 20$ | $x_{38} \leq 20$ | $x_{39} \leq 20$ | $x_{40} \leq 20$ | $x_{41} \leq 20$ | $x_{42} \leq 20$ |
| $x_{43} \leq 20$ | $x_{44} \leq 20$ | $x_{45} \leq 20$ | $x_{46} \leq 20$ | $x_{47} \leq 20$ | $x_{48} \leq 20$ | $x_{49} \leq 20$ |

Table 7: Upper bounds for variables $x_i$.

The above is actually an example of a transportation problem with a reasonable stepwise cost function; for more on that the reader is referred to [Michalewicz et al. (1990)] and [Vignaux & Michalewicz (1990)].

There are thirteen independent and one dependent equations here; therefore, we eliminate thirteen variables: $x_1, \ldots, x_8, x_{15}, x_{22}, x_{29}, x_{36}, x_{44}$. All remaining variables are renamed $y_1, \ldots, y_{36}$ preserving order, *i.e.* $y_1 = x_9$, $y_2 = x_{10}$, ..., $y_6 = x_{14}$, $y_7 = x_{16}$, ..., $y_{36} = x_{49}$. Each of these variables has to satisfy four two–sided inequalities, which result from the initial domains and our transformations. Now, each chromosome is a float vector $\langle y_1, \ldots, y_{36} \rangle$.

## 6.3 Experiments and results

For comparative experiments we planned on using a standard GA approach to constraints by penalties (see *e.g.* [Richardson et al. (1989)]) and a version (the *student version*) of GAMS, a

package for the construction and solution of large and complex mathematical programming models ([Brooke et al. (1988)]); we used MINOS version of the optimizer. However, we did not succeed in the former: the major issue in using penalty functions approach is assigning weights to the constraints — these weights play the role of penalties if a potential solution does not satisfy them. In experiments with the above problem the evaluation function *Eval* was composed of the optimization function $f$ and the penalty $P$:

$$Eval(\vec{x}) = f(\vec{x}) + P,$$

For our experiments we used a suggestion (see chardson et al. (1989)]) to start with relaxed penalties and to tight them as the run progresses. We used

$$P = k \cdot (\tfrac{t}{T})^p \cdot \overline{f} \cdot \textstyle\sum_{i=1}^{14} d_i$$

where $\overline{f}$ is the average fitness of the population at the given generation $t$, $k$ and $p$ are parameters, $T$ is the maximum number of generations, and $d_i$ returns the "degree of constraint violation". For example, for a constraint:

$$\textstyle\sum_{i \in W} x_i = val, W \subseteq \{1, \ldots, 49\},$$

and a chromosome $\langle v_1, \ldots, v_{49} \rangle$, the penalty (degree of constraint violation) $d_i$ was

$$d_i = |\textstyle\sum_{i \in W} v_i - val|.$$

We experimented with various values of $p$ (close to 1), $k$ (close to $\frac{1}{14}$, where 14 is total number of equality constraints; the static domain constraints were naturally satisfied by a proper representation), and $T = 8000$. However, this method did not lead to feasible solutions: in over 1200 runs (with different seeds for random number generator and various values for parameters $k$ and $p$) the best chromosomes (after 8000 generations) violated at least 3 constraints in a significant way. For example, very often the algorithm converged to a solution where the numbers on one diagonal were equal to 20 and all others were zeros:

$x_1 = x_9 = x_{17} = x_{25} = x_{33} = x_{41} = x_{49} = 20$
and $x_i = 0$ for others $i$.

As to GAMS, which only works with continuous functions, we reimplemented the problem using arc-tangent functions to approximate each of the five steps. A parameter $P_A$ was used to control the 'tightness' of the fit:

$$g(x_i) = c_i \cdot \begin{pmatrix} arctan(P_A(x_i - 2))/\pi + \frac{1}{2} & + \\ arctan(P_A(x_i - 4))/\pi + \frac{1}{2} & + \\ arctan(P_A(x_i - 6))/\pi + \frac{1}{2} & + \\ arctan(P_A(x_i - 8))/\pi + \frac{1}{2} & + \\ arctan(P_A(x_i - 10))/\pi + \frac{1}{2} & \end{pmatrix}$$

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_1 = 20.00$ | $x_2 = 0.00$ | $x_3 = 0.00$ | $x_4 = 1.93$ | $x_5 = 1.63$ | $x_6 = 1.47$ | $x_7 = 1.97$ |
| $x_8 = 0.00$ | $x_9 = 20.00$ | $x_{10} = 2.88$ | $x_{11} = 1.76$ | $x_{12} = 1.47$ | $x_{13} = 1.89$ | $x_{14} = 0.00$ |
| $x_{15} = 0.00$ | $x_{16} = 0.00$ | $x_{17} = 17.12$ | $x_{18} = 1.90$ | $x_{19} = 1.99$ | $x_{20} = 1.10$ | $x_{21} = 2.89$ |
| $x_{22} = 0.00$ | $x_{23} = 0.00$ | $x_{24} = 0.00$ | $x_{25} = 16.26$ | $x_{26} = 0.85$ | $x_{27} = 1.38$ | $x_{28} = 1.51$ |
| $x_{29} = 0.00$ | $x_{30} = 0.00$ | $x_{31} = 0.00$ | $x_{32} = 0.00$ | $x_{33} = 19.65$ | $x_{34} = 0.00$ | $x_{35} = 0.35$ |
| $x_{36} = 0.00$ | $x_{37} = 0.00$ | $x_{38} = 0.00$ | $x_{39} = 0.43$ | $x_{40} = 0.41$ | $x_{41} = 19.16$ | $x_{42} = 0.00$ |
| $x_{43} = 0.00$ | $x_{44} = 0.00$ | $x_{45} = 0.00$ | $x_{46} = 0.72$ | $x_{47} = 0.00$ | $x_{48} = 0.00$ | $x_{49} = 19.28$ |

Table 8: Solution found by our GA: $f(\vec{x}) = 24.15$.

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_1 = 20.00$ | $x_2 = 1.29$ | $x_3 = 0.95$ | $x_4 = 1.58$ | $x_5 = 1.52$ | $x_6 = 1.58$ | $x_7 = 0.08$ |
| $x_8 = 0.00$ | $x_9 = 18.71$ | $x_{10} = 0.39$ | $x_{11} = 1.59$ | $x_{12} = 1.58$ | $x_{13} = 0.12$ | $x_{14} = 5.61$ |
| $x_{15} = 0.00$ | $x_{16} = 0.00$ | $x_{17} = 18.66$ | $x_{18} = 1.56$ | $x_{19} = 1.47$ | $x_{20} = 1.59$ | $x_{21} = 1.72$ |
| $x_{22} = 0.00$ | $x_{23} = 0.00$ | $x_{24} = 0.00$ | $x_{25} = 18.27$ | $x_{26} = 1.25$ | $x_{27} = 0.00$ | $x_{28} = 0.48$ |
| $x_{29} = 0.00$ | $x_{30} = 0.00$ | $x_{31} = 0.00$ | $x_{32} = 0.00$ | $x_{33} = 19.47$ | $x_{34} = 0.53$ | $x_{35} = 0.00$ |
| $x_{36} = 0.00$ | $x_{37} = 0.00$ | $x_{38} = 0.00$ | $x_{39} = 0.00$ | $x_{40} = 0.00$ | $x_{41} = 20.00$ | $x_{42} = 0.00$ |
| $x_{43} = 0.00$ | $x_{44} = 0.00$ | $x_{45} = 0.00$ | $x_{46} = 0.00$ | $x_{47} = 0.71$ | $x_{48} = 1.18$ | $x_{49} = 18.11$ |

Table 9: Solution found by GAMS: $f(\vec{x}) = 96.00$.

This method let the system to find a feasible solution, but quite worse than those of our GA (Tables 8 and 9); the genetic algorithm found an optimal value 24.15 while the other system found one with 96.00.

The results indicate both usefulness of our method in the presence of many constraints and its superiority over some standard systems on nontrivial problems. Our modified genetic algorithm was run for 8000 iterations, with the population size equal forty. The parameters used in all runs are displayed in Figure 6. A single run of 8000 iterations took 2:28 CPU sec on Cray Y-MP.

GAMS was run on an Olivetti 386 with a math co-processor and a single run took 1:20 sec.

# 7 Conclusions

In this paper we discussed the use of genetic algorithms for numerical optimization problems. In particular, we concentrated on various modifications of classical genetic algorithms to overcome three major problems: handling of constraints, premature convergence, and local fine tuning. The preliminary results of several experiments are more than encouraging and suggest that the methods are very useful. They may lead towards solving some difficult

| Parameter | Value |
|---|---|
| pop_size | 40 |
| prob_mut$_{um}$ | .08 |
| prob_mut$_{bm}$ | .03 |
| prob_mut$_{nm}$ | .07 |
| prob_cross$_{sc}$ | .10 |
| prob_cross$_{sa}$ | .10 |
| prob_cross$_{wa}$ | .10 |
| $a$ | .25 |
| $b$ | 2.0 |

Figure 6: Parameters used for the $7 \times 7$ transportation problem: population size (pop_size), probability of uniform mutation (prob_mut$_{um}$), probability of boundary mutation (prob_mut$_{bm}$), probability of non-uniform mutation (prob_mut$_{nm}$), probability of simple crossover (prob_cross$_{sc}$), probability of single arithmetical crossover (prob_cross$_{sa}$), probability of whole arithmetical crossover (prob_cross$_{wa}$), coefficient $a$ for the whole arithmetical crossover, coefficient $b$ for $\Delta$ of the non-uniform mutation.

Operations Research problems. Currently, we are in the process of implementing a single system to incorporate all above ideas together. When completed, the system will be compared with many software optimization packages on different functions with nontrivial constraints. The system should be able to deal with very complex problems (thousands of variables, hundreds of constraints) since we need to represent only a relatively small population of potential solution vectors and apply new genetic operators to them.

Also, further extensions are planned to also handle discrete variables (integer, boolean, nominal) and to handle some classes of nonlinear constraints.

**Acknowledgments:** This research was supported by a grant from the North Carolina Supercomputing Center.

## References

[**Ackley, D.H. (1987)**] , *An Empirical Study of Bit Vector Function Optimization*, in [Davis (1987)], pp.170–204.

[**Antonisse, J. (1989)**] , *A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint*, in [Schaffer (1989)], pp.86–91.

[**Antonisse, H.J. & Keller, K.S. (1987)**] , *Genetic Operators for High Level Knowledge Representation*, in [Grefenstette (1987b)].

[**Bellman, R. (1957)**] , *Dynamic Programming*, Princeton University Press, Princeton, N.J.

[**Bethke, A.D. (1980)**] , *Genetic Algorithms as Function Optimizers*, Doctoral dissertation, University of Michigan.

[**Bertsekas, D.P. (1987)**] , *Dynamic Programming. Deterministic and Stochastic Models*, Prentice Hall, Englewood Cliffs, N.J.

[**Booker, L.B. (1982)**] , *Intelligent Behavior as an Adaptation to the Task Environment*, Doctoral dissertation, University of Michigan.

[**Booker, L.B. (1987)**] , *Improving search in genetic algorithms*, in [Davis (1987)], pp.61–73.

[**Brooke, A., Kendrick, D., Meeraus, A. (1988)**] , *GAMS: A User's Guide*, The Scientific Press.

[**Bosworth, J., Foo, N., Zeigler, B.P. (1972)**] , *Comparison of Genetic Algorithms with Conjugate Gradient Methods*, (CR-2093), Washington, DC: National Aeronautics and Space Administration.

[**Brindle, A. (1981)**] , *Genetic algorithms for function optimization*, Doctoral dissertation, University of Alberta, Edmonton.

[**Davis, L., (1987)**] (Editor), *Genetic Algorithms and Simulated Annealing*, Pitman, London.

[**DeJong, K.A. (1975)**] , *An analysis of the behavior of a class of genetic adaptive systems*, (Doctoral dissertation, University of Michigan).

[**DeJong, K.A. (1985)**] , *Genetic Algorithms: A 10 Year Perspective*, in [Grefenstette (1985)].

[**Dhar, V. & Ranganathan, N. (1990)**] , *Integer Programming vs. Expert Systems: An Experimental Comparison*, Communications of ACM, Vol.33, No.3, March 1990, pp.323–336.

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

[**Grefenstette, J.J. (1985)**] , Proceedings of the First International Conference on Genetic Algorithms, Pittsburgh, July 24–26, 1985, Lawrence Erlbaum Associates, Publishers.

[**Grefenstette, J.J. (1986)**] , *Optimization of Control Parameters for Genetic Algorithms*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-16, No.1, pp.122–128.

[**Grefenstette, J.J. (1987a)**] , *Incorporating Problem Specific Knowledge into Genetic Algorithms*, in [Davis (1987)], pp.42–60.

[**Grefenstette, J.J. (1987b)**] , Proceedings of the Second International Conference on Genetic Algorithms, MIT, Cambridge, July 28–31, 1987, Lawrence Erlbaum Associates, Publishers.

[**Groves. L., Michalewicz, Z., Elia, P., Janikow, C. (1990)**] , *Genetic Algorithms for Drawing Directed Graphs*, Proceedings of the Fifth International Symposium on Methodologies of Intelligent Systems, Knoxville, October 25–27, 1990, pp.268–276.

[**Holland, J. (1959)**] , *A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously*, Proceedinds of the 1959 E.J.C.C., pp.108–113.

[**Holland, J. (1975)**] , *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press.

[**Janikow, C. & Michalewicz, Z. (1990)**] , *Specialized Genetic Algorithms for Numerical Optimization Problems*, Proceedings of the International Conference on Tools for AI, Washington, November 6–9, 1990, pp.798–804.

[**Janikow, C. & Michalewicz, Z. (1991)**] , *Convergence Problem in Genetic Algorithms*, Fundamenta Informaticae, to appear.

[**Kirkpatrick, S., Gellatt, C., Vecchi, M. (1983)**] , *Optimization by Simulated Annealing*, Science, Vol. 220, No. 4598, p.671.

[**Laarhoven, P.J.M. van & Aarts, E.H.L., (1987)**] , *Simulated Annealing: Theory and Applications*, D. Reidell Publ. Comp.

[**Michalewicz, Z., Vignaux, G.A., Groves, L. (1989)**] , *Genetic Algorithms for Optimization Problems*, Proceedings of the 11-th NZ Computer Conference, Wellington, New Zealand, August 16–18, 1989, pp.211–223.

[**Michalewicz, Z., Vignaux, G.A., Hobbs, M. (1990)**] , *A Genetic Algorithm for the Nonlinear Transportation Problem*, ORSA Journal on Computing, Vol.3, No.4, 1991.

[**Michalewicz, Z. & Janikow, C. (1990)**] , *GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Constraints*, submitted to Communications of ACM.

[**Michalewicz, Z., Jankowski, A., Vignaux, G.A. (1990a)**] , *The Constraints Problem in Genetic Algorithms*, in *Methodologies for Intelligent Systems: Selected Papers*, (M.L. Emrich, M.S. Phifer, B. Huber, M. Zemankova, Z.W. Ras, Editors), Proceedings of the Fifth International Symposium on Methodologies of Intelligent Systems, Knoxville, October 25–27, 1990, pp.142–157.

[**Michalewicz, Z., Kazemi, M., Krawczyk, J., Janikow, C. (1990b)**] , *On Dynamic Control Problem*, Proceedings of the 29th IEEE Conference on Decision and Control, Honolulu, December 5–7, 1990.

[**Richardson, J.T., Palmer, M.R., Liepins, G., Hilliard, M. (1989)**] , *Some Guidelines for Genetic Algorithms with Penalty Functions* in [Schaffer (1989)].

[**Sasieni, M., Yaspan, A., Friedman, L. (1959)**] , *Operations Research Methods and Problems*, John Wiley and Sons.

[**Schaffer (1989)**] , Proceedings of the Third International Conference on Genetic Algorithms, June 4–7, 1989, Morgan Kaufmann Publishers.

[**Schaffer, J., Caruana, R., Eshelman, L., Das, R. (1989a)**] , *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, in [Schaffer (1989)], pp.51–60.

[**Sirag, D.J. & Weisser, P.T. (1987)**] , *Toward a Unified Thermodynamic Genetic Operator*, Proceedings of the Second International Conference on Genetic Algorithms, July 28-31, 1987, Lawrence Erlbaum Associates, Publishers.

[**Taha, H.A. (1982)**] , *Operations Research: An Introduction*, Collier Macmillan Publishers, London.

[**Vignaux, G.A. & Michalewicz, Z. (1989)**] , *Genetic Algorithms for the Transportation Problem*, Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems, Charlotte, NC, October 12–14, 1989, pp.252–259.

[**Vignaux, G.A. & Michalewicz, Z. (1990)**] , *A Genetic Algorithm for the Linear Transportation Problem*, IEEE Transactions on Systems, Man, and Cybernetics, Vol.21, No.2.

[**Von Neumann, J. (1966)**] , *Theory of Self-Reproducing Automata*, edited by Burks, University of Illinois Press.

[**Ulam, S. (1949)**] , *On the Monte Carlo Method*, Proceedings of the 2nd Symposium on Large Scale Digital Calculating Machinery, Harvard University Press, pp.207–212.