BASIC—A genetic algorithm for engineering problems solution

Elisaveta G. Shopova *, Natasha G. Vaklieva-Bancheva

Institute of Chemical Engineering, Bulgarian Academy of Sciences, "Acad. G. Bonchev" street Bl.103, 1113 Sofia, Bulgaria

Received 12 August 2005; received in revised form 27 January 2006; accepted 9 March 2006

Abstract

This paper introduces in details a genetic algorithm-called BASIC, which is designed to take advantage of well known genetic schemes so as to be able to deal with numerous optimization problems. BASIC GA follows all common steps of the genetic algorithms. It involves real representation schemes for both real and integer variables. Three biased selection schemes for reproduction; four for recombination and three for mutation are applied in it and a new selection scheme for replacement is approached.

BASIC GA can be easy adjusted to the concrete problems by fitting its global and local parameters. It provides an opportunity to the genetic operators to be extended with new schemes.

A range of various optimization problems has been solved to test its capability. To handle all sorts of constraints the static and dynamic penalty functions are used. The solutions obtained are commensurable with other genetic algorithms and solution techniques.

© 2006 Elsevier Ltd. All rights reserved.

Keywords: Genetic algorithm; Real representation; Selection; Recombination; Mutation; Replacement; Optimization problems

1. Introduction

A large scale of design, control, scheduling or other engineering problems results in solution of optimization problems. Different classes of search techniques like calculus-based techniques, guided random search techniques and enumerative techniques are developed to deal with the optimization. Recently, evolutionary programming and genetic algorithms (GAs), as a part of it, are successfully used for solution of a large range of optimization problems.

Many engineering and chemical engineering problems are solved by employing genetic algorithms. Carrillo-Ureta, Roberts, and Becerra (2001) have used it to optimize batch beer fermentation. Genetic algorithm is set up to get the best feeding profile for minimization of the reaction time with a temperature constraint in fed-batch reactor (Nougues, Grau, & Puigjaner, 2002). The influence of the internal GAs parameters on its performance has been studied on the large scale polymer design problem (Sundaram & Venkatasubramanian, 1998). Azzaro-Pastel, Bernal-Haro, Baudet, Domenech, and Pibouleau (1998) have proposed a two-stage methodology comprising discrete-event simulation model and genetic algorithm for solving the job-shop scheduling problems. Aiming to explore the power of stochastic search methods for solution of challenging problem in experimental physics, Barash, Orel, and Vemuri (2000) also have applied the genetic algorithm. Choy and Sanctuary (1998) have used it for estimation of NMR parameter for quantitative signal analysis, etc. However, in all of the mentioned studies the genetic algorithms are created to solve the concerned problems. We have not found in the open, for us, literature developments faced to solution of a large scale of engineering problems, which was a reason to decide such genetic algorithm, easy adjustable to different engineering problems, to be designed.

The aim of this paper is to present BASIC—the genetic algorithm developed in the Bulgarian Academy of Sciences, Institute of Chemical Engineering. It is designed to exploit the benefit of the existing genetic schemes so as to be able to deal with various engineering problems. It uses real representation schemes both for real and integer variables, includes number of selection, reproduction and mutation schemes in respective genetic operators, innovates a replacement scheme and can be easily fitted to the concrete problems. The solutions obtained by BASIC GA are commensurable with such of other GAs, which is tested on the number of examples.

* Corresponding author. Tel.: +359 2 979 3481; fax: +359 2 979 3481.
E-mail address: eishopova@mail.bg (E.G. Shopova).

0098-1354/$ – see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compchemeng.2006.03.003

CACE-3228; No. of Pages 17
This paper is organized as follows: Section 2 covers the basic features of the genetic algorithms as an optimization technique. The BASIC genetic algorithm is considered in detail in Section 3. Section 4 provides testing examples. Final conclusions are given in Section 5.
2. Basic features of genetic algorithms

2.1. Terminology

Genetic algorithms (GAs) are stochastic techniques whose search methods model a natural evolution. That is why the terminology used in GAs is taken from biology, which is listed in Table 1.

2.2. General description of genetic algorithms

The genetic algorithms start with randomly chosen parent chromosomes from the search space to create a population. They work with chromosome genotype. The population “evolves” towards the better chromosomes by applying genetic operators modeling the genetic processes occurring in the nature—selection, recombination and mutation.

The recombination is carried out after selection process is finished. It combines, with predefined probability, the features of two selected parent chromosomes forming similar children.

After three operators are carried the offspring is inserted into the population, replacing the parent chromosomes in which they were derived from, producing a new generation. This cycle is performed until the optimization criterion is met.

3. BASIC genetic algorithm

BASIC follows the described above common steps of the genetic algorithms. Its architecture is presented by the block scheme shown in Fig. 1. It works with a predefined constant size of population.

At the first generation, BASIC initializes a population of randomly created individuals. Applying morphogenesis functions their phenotypes are determined. Afterwards they are used to calculate the values of the objective function and to determine respective fitness functions.

At the next steps the evolutionary operators take place to create the offspring. Firstly, a biased selection for reproduction is carried out. The algorithm operates with the fitness function values to provide the most prospective samplings for a crossover. They gather in a sampling pool. Then, the samplings go to recombination. Randomly chosen individuals form couples, whose number is equal to the number of samplings. If, for a given couple, a predefined crossover probability has happened, their chromosomes recombine providing two children, otherwise, parents pass directly in the offspring. Finally, the mutation takes place. Each gene of each offspring’s chromosome goes to mutation if a predefined mutation probability has happened.

In the last stage, selection for replacement takes place to produce a new population for the next generation. The offspring decode to obtain the respective solutions. Both, the parents’ and children’s chromosomes collect in a replacement pool. The elite individual, corresponding to the best solution in the pool passes in the new population. Further, selection goes unbiased randomly drawing chromosomes from the pool till next population is completed. At the end of this stage, the number of generation increases.

BASIC uses the generation number as a stop criterion. It checks for the stop criterion fulfillment. If it is met, the obtained best solution is proposed as a problem solution. In the opposite case, the loop is closed through the fitness functions calculation for the new population.

BASIC GA is coded by using MATHCAD 7 program’s tools. Aiming to make it more adaptive to various optimization problems we involved a different number of schemes in the genetic operators. Thus, the selection for recombination comprises three schemes; recombination—four; and mutation—three. Moreover, as it can be seen from Fig. 1, two types of ranking schemes are included in the rank-based selection, which additionally increases BASIC flexibility. Furthermore, it is design with an open architecture providing an opportunity to be extended with new genetic schemes. Only one scheme for each evolutionary operator has to be chosen running BASIC GA.

BASIC GA can be easy adjusted to the concrete problems by fitting its parameters. Later are divided to two groups—global and local parameters. The global parameters as: the population size, number of generation, number of samplings and crossover and mutation probabilities, concern the whole algorithm and do not depend on the chosen schemes of genetic operators. The local parameters refer to some of schemes involved. They aim to regulate respective schemes and will be pointed below in their more detailed consideration.

3.1. Representation

All genetic algorithms start with initialization of the first generation. They operate with chromosomes’ genotypes, which represent the solutions (phenotypes) coded into strings of bits of...
a given length. Binary, integer or real search space can be used for genes representation. The morphogenesis function associates each genotype with its phenotype. Trying to face BASIC GA to larger scale of engineering problems we chose the continuous search space, using real representation schemes for both, the real and integer variables. The idea came from the research of Oyama, Obayashi, and Nakahashi (2000), which have proposed a sophisticated approach, named dynamic coding, to dynamically alter the coarseness of the search space.

3.1.1. Dynamic real representation

We have applied the Oyama’s technique to treat the continuous design space. Using the boundaries of the real
where each integer \( n \) is a part of this space, with a length of the integer variables:

\[
x_m = (\min - \max) \cdot \text{gene}_m + \max.
\]

Thus, the genes presented as real strings are:

\[
x_m \in [0, 1).
\]

3.2. Fitness evaluation

Once the genetic representation is defined, the next step is to determine the fitness functions associated to the solutions. They are calculated on the base of the chromosomes phenotype. A fitness function measures a distance between good and bad solutions by mapping the solution to a non-negative interval.

Following the Handbook of Evolutionary Computation (1997), two mappings, corresponding to the minimization and maximization problems are included in BASIC GA:

(i) For minimization problems, the fitness function corresponding to some solution \( i \) calculates according to the equation:

\[
\text{fitness}(\vec{x}_i(t)) = \frac{1}{1 + f(\vec{x}_i(t)) - f_{\text{max}}(t)},
\]

where \( X \) is a design space of the vector of control variables \( \vec{x} \); and \( f_{\text{max}} \) is the minimum observed value of the objective function up to generation \( t \).

(ii) For maximization problems it determines by a following relation:

\[
\text{fitness}(\vec{x}_i(t)) = \frac{1}{1 + f_{\text{max}}(t) - f(\vec{x}_i(t))},
\]

where \( f_{\text{max}} \) is the maximum observed value of the objective function up to generation \( t \).

Using of different mappings for minimization and maximization problems requires the type of the optimization problem to be indicated in BASIC GA before its starting.

3.3. Selection

Selection is the first genetic operator taking place in BASIC GA. Generally, two types of selection apply in the genetic algorithms: selection for reproduction and selection for replacement. First determines the samplings that will produce offspring, while the second provides next population. Both will be discussed in this section.

3.3.1. Selection for reproduction

This kind of selection aims to identify better chromosomes of the population that would be taken for reproduction. In BASIC GA, the identification carries out in compliance with the fitness functions values. Last are used to assign a selection probability to each individual.

BASIC GA includes three biased selection schemes for reproduction: roulette-wheel, rank based, and tournament. All of them are replicas of these presented in the Handbook of Evolutionary Computation (1997). The number of samplings selected for reproduction is a global BASIC parameter. Selected individuals are collected in a sampling pool.
3.3.1. Roulette-wheel selection. Roulette-wheel is the simplest proportionate selection scheme. Its operation is illustrated in Fig. 2.

According to this scheme, the individuals of the population assume as slots of the Roulette-wheel. Each slot is as width as the probability for selection of the corresponded chromosome is great. The scaled fitness function is used to calculate respective selection probabilities:

\[ P_r(i) = \frac{\text{fitness}(\vec{x}_i)}{\sum_{j=1}^{\mu} \text{fitness}(\vec{x}_j)} \]

(11)

where \( \mu \) is the size of the population.

Roulette-wheel selection is emphasized to the better individuals in the population and exerts a large pressure on the search process. The expected number of copies in the sampling pool that some solution could receive is proportional to its selection probability. This could cause loosing of genetic diversity and could lead to a premature convergence to a suboptimal solution.

Drawing the samplings for reproduction is carried by applying of the so called Roulette-wheel sampling algorithm. The procedure repeats till the sampling pool is completed and is shown below:

begin
chrom ← 1;
sum ← Pr(\text{chrom});
a ← random number (0, 1);
while sum ◁ a do
begin
chrom ← chrom + 1;
sam ← sum + Pr(\text{chrom});
end;
sampling ← chrom
end.

3.3.1.2. Rank-based selection. Rank-based selection is a similar to the proportionate selection schemes. In it, individual’s rank, instead of the fitness function, is used to calculate the selection probability. Last gives a better chance of chromosomes with small fitness values to take part in the reproductive process and preserves populations from a premature convergence.

Applying this selection scheme in BASIC GA, we arrange the individuals in ascending order according to the value of their fitness functions. Afterwards, each chromosome obtains a rank depending on the place it has taken in this order. Thus, the worst chromosome, which is first in the order, has a rank 0, while for the best one, which is the last—the rank is \( (\mu - 1) \).

Generally, the selection probabilities could be calculated using different types of ranking. BASIC GA comprises two rankings: linear and square. Passing on each of them a selective pressure \( \beta \) is used to calculate the selection probabilities. It is set as a local BASIC parameter and defines before respective ranking calls.

(i) Linear ranking. According to it, the selection probability is proportional to the rank of each individual and calculates as follows:

\[ P_{\text{RankLin}}(i) = a + \frac{\text{rank}(i) / (\mu - 1) \beta}{\mu} (\beta - a) \]

(12)

In the Eq. (12) the selective pressure \( \beta \) presents the expected number of offspring to be allocated to the best individual, while \( a \) refers to the worst one. In the Handbook of Evolutionary Computation (1997) is proved that \( \beta \) must range in the interval:

\[ 1 \leq \beta \leq 2 \]

(13)

When \( \beta \) is equal to 1 the selection probability for all individuals gets an equal value. If it raises from 1 to 2 the selection probability of the individuals of higher ranks increases while this of smaller decreases. Thus the selection becomes more biased and exerts a larger pressure on the final solution. In BASIC GA, the local parameter \( \beta \) can be varied in the pointed interval (Eq. (13)). If it is not set, linear ranking calls \( \beta = 2 \) as default. The parameter \( a \) calculates according to the equation:

\[ a = 2 - \beta \]

(14)

(ii) Square ranking. Square ranking is a simplest case of non-linear rankings. The assigned to each individual selection probability calculates on the base of the square of its rank:

\[ P_{\text{RankSqu}}(i) = a + \frac{\text{rank}(i)^2 / (\mu - 1)^2 \beta}{c} (\beta - a) \]

(15)
In Eq. (15), except the selective parameters $\beta$ and $\alpha$, a normalization factor $c$ is used:

$$c = (\beta - \alpha)\mu(2\mu - 1)/(6\mu - 1) + \mu\alpha.$$  

(16)

The local parameter $\beta$, in the square ranking, is changed in the same boundaries as for the linear ranking. It plays the similar role as in the linear, but makes the selection less biased to the individuals of higher ranks. Taking into account that the selection probabilities must range from $\alpha/c$ to $\beta/c$, the parameter $\alpha$ is arbitrary choosing in the boundaries:

$$0 < \alpha < \beta.$$  

(17)

Both rank selections, linear and square, are the most unbiased selection schemes involved in BASIC GA.

Using thus calculated selection probabilities, the samplings for reproduction are drawing by applying the same Roulette-wheel sampling algorithm as in the roulette-wheel selection.

3.3.1.3. Tournament selection. This selection type models the tournament games. Its principle is illustrated in Fig. 3. It uses the fitness function values for evaluating the prospective for reproduction chromosomes.

According to the tournament selection, randomly chosen, from the population, $q$ individuals formed a tournament size, call to a contest. Only one individual that has the highest fitness value is the winner of the game. Its chromosome copies in the sampling pool. The tournament repeats till the sampling pool is complete.

The tournament size is set as a local parameter in BASIC GA. Its value can vary from two to the population size $\mu$. Value 2 is used as default. Tournament selection gets so much biased as the tournament size gets larger.

Generally, selection produces an effect on the BASIC GA convergence. When we choose a specific selection scheme one must be taken into account the pressure that it exerts on process of searching and on the selected solutions. Depending on the chosen selection scheme the appropriate crossover and mutation operators could be set. Thus the algorithm could be preserved from a premature convergence and the population keeps its variety as long as the search process progresses.

3.3.2. Selection for replacement

Selection for replacement creates the new generation from the current one and the obtained offspring. We have approached a new selection scheme for replacement when the unbiased selection is combined with the elitism. Its operational scheme is illustrated in Fig. 4.

Firstly, the obtained offspring is decoded and respective solutions are determined. Then, the offspring solutions joint to the parents’ thus forming a common pool for replacement. The elite individual corresponding to the best solution in this pool is determined and copied in the new generation. The rest chromosomes, completing the generation, are determined stochastically, with uniform distribution.

Applying the proposed selection for replacement we achieve the following:

(i) keep the obtained up to some generation $t$ best solution, and
(ii) preserve the population from the premature convergence.

In principle, number of copies of the elite individual could be greater then one. However, our experience shows that its
increasing affects on the population convergence. This is the reason to accept only one copy of the elite individual in proposed selection for replacement.

3.4. Crossover

Crossover is the second genetic operator called in BASIC GA. Generally, it aims to produce offspring by recombination the chromosomes of two individuals, named parents. Crossover exploits the search space determined by the parents.

BASIC GA is designed to get two children of two parents. For that purpose, two individuals randomly choose from the sampling pool form couples. Number of couples is equal to the pool size. The proportion of parents undergoing the crossover controls by the crossover probability. It determines how frequently the crossover operator invokes. If, for a given couple, the crossover probability has happened their chromosomes are recombined and produce children, otherwise they are directly copied in the offspring. Thus, the size of offspring obtained is twice of the sampling pool.

BASIC GA includes four recombination schemes: N-points crossover; uniform crossover; arithmetic crossover and blend crossover. First three are replicas of the same schemes described in the Handbook of Evolutionary Computation (1997), while the fourth is designed following the approach proposed by Eschelman and Schaffer (1993).

Crossover probability is established as a global BASIC GA parameter. Its value may be set in the interval [0, 1]. BASIC GA takes it equal to 0.75 as default.

3.4.1. N-point crossover

N-point crossover and especially its particular cases of one and two point’s crossover is the simplest discrete recombination scheme.

According to it, firstly, N randomly chosen, with uniformly distribution breakpoints (or crossover points) are located on the parents’ chromosomes. They delineate the chromosomes segments for exchanging. Then, respective segments exchange and after combining produce two new individuals (children).

An illustration for recombination in two points, k1 and k2, is shown below:

\[
\begin{align*}
\text{Parents} & : \\
\text{chrom}1 & = \text{gene}_{11} \text{ gene}_{12} \cdots \text{gene}_{1k1-1} \cdots \text{gene}_{1k_1} \text{ gene}_{1k_1+1} \cdots \text{gene}_{1m} \\
\text{chrom}2 & = \text{gene}_{21} \text{ gene}_{22} \cdots \text{gene}_{2k1-1} \cdots \text{gene}_{2k_1} \text{ gene}_{2k_1+1} \cdots \text{gene}_{2m} \\
\text{Offspring} & : \\
\text{chrom}1' & = \text{gene}_{11} \text{ gene}_{12} \cdots \text{gene}_{2k1} \cdots \text{gene}_{2k_1+1} \cdots \text{gene}_{1m} \\
\text{chrom}2' & = \text{gene}_{21} \text{ gene}_{22} \cdots \text{gene}_{1k1} \cdots \text{gene}_{1k_1+1} \cdots \text{gene}_{2m} 
\end{align*}
\]
In BASIC GA, number of crossover points is set as a local parameter. Its value must be less than the number of control variables for the problem under solution. As default, one crossover point is accepted.

3.4.2. Uniform crossover

Uniform crossover also belongs to the discrete recombination schemes. According to it, each gene of each child is created by randomly selecting respective gene from one of both parents. Both parents have an equal chance to contribute in the children’s chromosomes. An illustration of uniform crossover is shown below:

- Parents
  - chrom1 = gene11, gene12, gene13, gene14, ..., gene1M
  - chrom2 = gene21, gene22, gene23, gene24, ..., gene2M

- Offspring
  - chrom1′ = gene11, gene12, gene23, gene14, ..., gene1M
  - chrom2′ = gene11, gene12, gene23, gene14, ..., gene2M

3.4.3. Arithmetical crossover

In the contrast of the previous two crossover schemes the arithmetical crossover exploits points of the search space belong to a line connecting both parents. It is very functional for real representation.

Arithmetical crossover defines as a linear combination of two vectors. If chrom1 and chrom2 are parents’ chromosomes subject to recombination, according to this approach respective offspring’s chromosomes determine following next equations:

chrom1′ = ω · chrom1 + (1 − ω) · chrom2
chrom2′ = ω · chrom2 + (1 − ω) · chrom1′

where the parameter ω, is randomly choosing in the interval [0, 1].

3.4.4. Blend crossover

Blend crossover is most common recombination approach in case of real representation. According to it, the offspring generates on the space segment defined by two parents and extended equally on both sides by using a specified parameter ε. Thus, if chrom1 and chrom2 are parents’ chromosomes the children’s chromosomes express as:

chrom1′ = ξ · chrom1 + (1 − ξ) · chrom2
chrom2′ = (1 − ξ) · chrom1 + ξ · chrom2′

where ξ determines as follows:

ξ = (1 + 2ε) · a − ε.

3.5. Mutation

Mutation is the third evolutionary operator for BASIC GA. Its general aim is to create a new individual from only one chromosome by changing one or more genes in it. Each gene of each individual is possible subject to mutation. The sense of mutation is to explore the search space. In BASIC GA we accept a police to mutate only the offspring.

The process controls by the mutation probability. It determines the mutation frequency. The mutation probability is set as a global BASIC parameter. Its value may vary in the interval [0, 1]. However, it is recommended to be a small number. Value 0.1 is accepted as default in BASIC GA. If it is set to 0, the mutation operator omits and created after crossover offspring recognizes as the final.

BASIC GA involves three mutation schemes: uniform, non-uniform and breeder mutation. First is adopted, for the real representation, the approach described in the Handbook of Evolutionary Computation (1997) for binary strings; the second duplicate the methodology proposed by Michalewicz, Logan, and Swaminathan, (1994) and Michalewicz (1996); while the third is designed according to description presented by Choy and Sanctuary (1998).

3.5.1. Uniform mutation

The uniform mutation is a simple mutation scheme. According to it, at the beginning, the positions of the genes that would be mutated are determined. All genes have an equal chance for mutation but, only these, for which the mutation probability has happened, undergo. After that, new genes create to replace the selected. They are randomly drowned with uniform distribution from the search space [0, 1].

The uniform mutation is a quite disruptive mutation operator, which is appropriate to be combined with biased selection and not so disruptive crossover schemes. When we choose such of mutation scheme, it is recommended a low mutation probability to be set.

3.5.2. Non-uniform mutation

The non-uniform mutation is a type of zero-mean mutations. The common form of zero mean mutation is:

gene′ = gene + Mu,

where Mu is a global BASIC parameter. Its value must be chosen in the interval [0, 0.5]. Value 0.5 is used as default. At ε = 0 the blend crossover approaches arithmetical one. When ε arises from 0 to 0.5 the values of children’s genes could be assigned not only within the interval defined by the parents’ genes but out of it. Last gives an additional variety to the formed offspring.

Arithmetical and especially blend crossover are considerably disruptive crossover operators, and could be combined with more biased selection schemes. They cause for saving the variety in the population and keep out of premature convergence. Both are appropriate for solution of different types of problems such as NLP, MNLP, LP, while two discrete crossover schemes (N-points and uniform) have a good behavior in problems with binary variables.
where \( M \) is a random variable chosen such that the expected difference between both, non-mutated and mutated genes to be zero.

According to the non-uniform mutation, the search space of gene mutation strongly constricts with increasing of generations. Thus, the value of the mutated gene determines following next equation:

\[
\text{gene}_{m}(t) = \begin{cases} 
\text{gene}_{m}(t) + \Phi(t, \gamma) & \text{if } u < 0.5 \\
\text{gene}_{m}(t) - \Phi(t, \gamma) & \text{if } u \geq 0.5
\end{cases},
\]

where \( \text{gene}_{m}(t) \) is the \( m \)th gene of the chromosome up to generation \( t \) and \( \text{gene}_{m}(t) \in [0,1] \); \( u \) is a random uniform number \( u \in [0, 1] \).

The function \( \Phi(t, \gamma) \) calculates according to:

\[
\Phi(t, \gamma) = y\left(1 - \frac{r}{T}\right)^{b} \\ y = \begin{cases} 
1 - \text{gene}_{m}(t) & \text{if } u < 0.5 \\
\text{gene}_{m}(t) & \text{if } u \geq 0.5
\end{cases},
\]

where \( T \) is the maximum generation number; and \( b \) is a parameter chosen by the operator to determine the degree of non-uniformity.

In BASIC GA \( b \) is set equal to 1.

### 3.5.3 Breeder mutation

The mutation of the gene under breeder mutation scheme calculates as follows:

\[
\text{gene}_{m}(t) = \text{gene}_{m} \pm \text{range} \cdot \theta,
\]

where signs + and − are with equal probability; range is 0.5 of the domain of variable (in case of real presentation the domain is \([0, 1]\)) and:

\[
\theta = \sum_{m=1}^{M} a(m) 2^{-(m-1)} \quad \text{where}
\]

\[
a(m) = \begin{cases} 
1 & \text{with probability} \frac{1}{M} \\
0 & \text{otherwise}
\end{cases}.
\]

### 3.6 Parameters settings

BASIC GA is faced to a range of engineering problems. However, its behavior depends on, from one side, the proper setting of global and local parameters and combination of genetic operators and from the other, the particular problem under solution. The global BASIC GA parameters as the population size, number of samplings and crossover and mutation probabilities affect on the algorithm convergence. Thus, the large size of population decreases the speed of convergence, but keeps the population variety and algorithm could trap in a local optimum of less probability. On contrary, a small size of population easier leads to a premature convergence. Commonly, it is accepted the population size to be 10 times greater then the number of variables.

BASIC GA works well when the population size is roughly \( 20 \times (30 + \text{variable numbers}) \times 10 \). The number of samplings also causes a similar affect on the algorithm convergence as the population size. Our experience shows that a good convergence could be achieved if the number of samplings is approximated to half of population size. Concerning mutation probability, the accepted practice is to evaluate it around to \( 1/\text{variable numbers} \), while the crossover probability usually is set around \( 0.7–0.8 \).

Generally, the selection has a largely influence on the algorithm convergence. In BASIC GA we have accepted the policy of biased selection for reproduction and unbiased, but keeping the elitism, for replacement. As it was pointed in the end of Section 3.3.1, choosing some selection for reproduction, we must take into consideration its biasness. The most prejudiced scheme in BASIC GA is the roulette-wheel. To avoid the premature convergence it is recommended to combine it with more disruptive crossover and mutation operators or to increase the crossover and mutation probabilities. The biasness of rank selection, both linear and square, could be controlled through the local BASIC GA parameter \( f \), while for the tournament selection it increases with increasing of the tournament size, which is also set as a local parameter. The created from us and used in BASIC GA selection for replacement, also in a large extent, preserves the algorithm from the premature convergence.

The good performance of BASIC GA requires the proper choice of crossover and mutation operators. Except on the used selection scheme, their picking depends on the type of the problem. Thus, N-points and uniform crossovers deal better with binary problems, while arithmetical and blend crossovers with NLP, MNLP and LP. Arithmetical and blend crossovers and uniform mutation are most disruptive genetic operators involved in BASIC GA, which strongly affect on keeping of the population diversity. Additionally, in blend crossover the variety of the population is affected on the setting of the local parameter \( e \).

Once again we would underline that the discovery of optimal settings for both the global and local BASIC GA parameters and genetic operators depends on the study under consideration. But, in general, their settings have to:

(i) keep the diversity of the population in solution progress;
(ii) preserve the algorithm from the premature convergence; and,
(iii) lead to obtaining the best solutions for the reasonable number of generations.

### 3.7 Constrained problems

The genetic algorithms are direct methods that deal only unconstrained problems. To handle all sorts of constraints (linear, non-linear, equality and inequality) the constraint-handling techniques have been adopted over the years. Special representations and operators are proposed by Davis (1991), Beanie (1992, 1994), Michalewicz (1992), Kowalczuk (1997), Glover (1977), and Dasgupta and Michalewicz (1997). Repair algorithms, separation of constraints and objectives are discussed by Paredis (1994), Deb (2000), Powell and Skolnick (1993), Schoenauer and Xanthakis (1993), Garis (1990), Fonseca and Fleming...
where $j$ is a small number.

$x$ based on length of search, a dynamic penalty function. It incorporates a dynamic aspect penalty functions that overcomes many of these difficulties is presented below.

Adeli and Cheng (1994), Kim and Myung (1997), Myung and Kim (1998), and Powell (1969) have regarded different kinds of hybrid methods, etc.

Aiming to transform the constrained optimization problems to the unconstrained ones the most common way is application of well developed over the years penalty techniques (Courant, 1943; Carroll, 1961; Fiacco & McCormick, 1968). We also faced to the penalty techniques both the static and dynamic penalizations, as it is proposed by Bean and Hadji-Aoulos (1992), to handle constrained optimization problems using BASIC GA.

Static penalty function is a simple method to penalize solutions that violate, based only on the number of constraints violated. The defined in BASIC GA static penalty function is:

$$f_{p}(x) = f(x) + \sum_{j=1}^{n} C_j \delta_j$$

where $\delta_j = 1$ if constraint $j$ is violated

$$\delta_j = 0$$

if constraint $j$ is satisfied

(30)

where $j$ is a number of constraint; $f(x)$ is the penalized objective function; $f(x)$ is the unpunished solution, and $C_j$ is a constant imposed for violation of constraint $j$.

Usually, we pose the penalty coefficients $C_j$ to be equal for all constraints. Moreover, we choose them to be at the same order, or higher than the order of the objective function. For the purpose we assess the order of the objective function using the boundary values of control variables.

The static penalty function applies successfully for solution of the different problems, but the primary deficiency is that the coefficients $C_j$ are set roughly. A variation of distance-based penalty functions that overcomes many of these difficulties is a dynamic penalty function. It incorporates a dynamic aspect based on length of search, $t$. In BASIC GA we apply the dynamic penalty function formulated in the following manner:

$$f_{p}(x) = f(x) + \sum_{j=1}^{n} C_j \delta_j$$

$$d_j = \begin{cases} 
0 & \text{if } |g_j(x)| \leq \Delta, |h_j(x)| \leq \Delta \\
|g_j(x)| & \text{otherwise}
\end{cases}$$

(31)

where $d_j$ is a distance metric of constraint $j$ applied to solution $x$, $g_j(x)$ and $h_j(x)$ are inequality and equality constraints and $\Delta$ is a small number.

For $x(t)$ we exploit the following monotonic and non-decreased function in value with $t$:

$$x(t) = 1 + B \Delta t$$

(32)

Metric for $t$ is a number of generations, while $B$ is a parameter set by the user $0 \leq B \leq 1$.

The coefficients $C_j$ in this formulated dynamic penalization are set in the same way as the static penalty coefficients.

Using thus formulated penalty functions both, static and dynamic lead to successive results for the tested models. Although, these penalty functions typically require problem specific tuning to perform well.

4. Genetic algorithm testing

BASIC GA is tested on a number of examples used in the literature for genetic algorithms testing. Ten of them are presented below as an illustration. They are from MINLP, NLP and binary programming. Problems’ details are shown in Table 2 . Each problem was run at least 10 times with different seed. All simulations were done on Pentium IV machine. A possible comparison is made with solutions obtained by existing algorithms. For the most of problems the algorithm gave commensurable results. Table 3 presents the BASIC GA performance.

4.1. Test problem 1

It is a MINLP minimization problem taken from Floudas, Aggarwal, and Ciric (1989). It has four binary and three continuous variables with nine inequality constraints. The obtained optimum solution with BASIC GA is 4.971699587. It is equal to reported as a best solution found by Deb’s method. Summanwar has obtained the best solution of 4.57959 with modified constraints and definitely more complicated algorithm. BASIC GA reaches best result at the low number of generations, only 1000.

4.2. Test problem 2

It is an NLP minimization problem taken from Summanwar et al. (2002). It has five continuous variables and six inequality constraints. Deb solved this problem with genetic algorithms employing penalty method without penalty coefficients. In his procedure he also incorporated special niching schemes to avoid premature convergence to local optima. We have obtained with BASIC GA the best solution of −30665.493151175 and our solution is a commensurable than reported from Deb’s (2000) and Summanwar et al. (2002) ones. Applying non-linear mutation we obtain better result than the best-known solution.

4.3. Test problem 3

It is minimization problem also taken from Summanwar et al. (2002). This is a NLP minimization problem with 10 continuous variables and eight inequality constraints. The global optimum of the problem is 24.30620. Deb has reported the best solution as 24.372. Our best solution is approximately around this value.

4.4. Test problem 4

It is a constrained handling methods used to solve this problem have
### Problem details

<table>
<thead>
<tr>
<th>No.</th>
<th>Problem description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sources of the example—Floudas et al. (1989) and Summannar et al. (2002)</td>
</tr>
<tr>
<td></td>
<td><strong>Objective function</strong></td>
</tr>
<tr>
<td></td>
<td>$\min \ y_1 + y_2 + \sum_{i=3}^{10} (y_i - 100y_i^2 + 10y_i) + 5y_{10} - 15y_{10}^2 + 2y_{10}^3 - 0.5y_{10}^4$</td>
</tr>
<tr>
<td></td>
<td><strong>Constraints</strong></td>
</tr>
<tr>
<td></td>
<td>$-10y_1 + 4y_2 + 17y_3 - 2y_4 = 0$</td>
</tr>
<tr>
<td></td>
<td>$3y_1 - 2y_2 - 5y_3 + 2y_4 = 0$</td>
</tr>
<tr>
<td></td>
<td>$-3y_1 - 2y_2 - 5y_3 + 3y_4 + 11y_5 + 2y_6 + 15y_7 + 7y_8 - 3y_9 = 0$</td>
</tr>
<tr>
<td></td>
<td>$0 \leq y_i \leq 10$, $i = 1, \ldots, 10$</td>
</tr>
</tbody>
</table>

| 2   | Source of the example—Summannar et al. (2002) |
|     | **Objective function** |
|     | $\min x_1 + x_2 + x_3 - 14x_1 - 10x_2 + 10x_3 - 10x_4 + 4x_5^2 + 2x_6^2 - 2x_7^2 + 5y_{10}^2 + 7x_9 - 11y_1 - 12y_2 + 8y_3 + 2y_4 + 7y_5 + y_6 + 13y_7 + 12y_8 + 9y_9$ |
|     | **Constraints** |
|     | $-10x_1 + 4x_2 + 17x_3 - 2x_4 = 0$ |
|     | $3x_1 - 2x_2 - 5x_3 + 2x_4 = 0$ |
|     | $-3x_1 - 2x_2 - 5x_3 + 3x_4 + 11x_5 + 2x_6 + 15x_7 + 7x_8 - 3x_9 = 0$ |
|     | $0 \leq x_i \leq 10$, $i = 1, \ldots, 10$ |

### Table 2 (Continued)

<table>
<thead>
<tr>
<th>No.</th>
<th>Problem description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Sources of the example—Michalewicz (1995) and Deb (2000)</td>
</tr>
<tr>
<td></td>
<td><strong>Objective function</strong></td>
</tr>
<tr>
<td></td>
<td>$\min x_1 + x_2$</td>
</tr>
<tr>
<td></td>
<td><strong>Constraints</strong></td>
</tr>
<tr>
<td></td>
<td>$1 - 0.0025\sin(x_1 + x_2) \geq 0$</td>
</tr>
<tr>
<td></td>
<td>$1 - 0.01\sin(x_1 + x_2) \geq 0$</td>
</tr>
<tr>
<td></td>
<td>$x_1 - 83.33332x_2 \geq 1000$, $x_1 + x_2 \geq 1000$</td>
</tr>
<tr>
<td></td>
<td>$100 \leq x_1 \leq 10000$</td>
</tr>
<tr>
<td></td>
<td>$10 \leq x_2 \leq 10000$, $r = 4, \ldots, 8$</td>
</tr>
</tbody>
</table>

### Table 2 (Continued)

<table>
<thead>
<tr>
<th>No.</th>
<th>Problem description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Source of the example—Michalewicz (1995) and Deb (2000)</td>
</tr>
<tr>
<td></td>
<td><strong>Objective function</strong></td>
</tr>
<tr>
<td></td>
<td>$\min x_1 + x_2 + x_3 - 14x_1 - 10x_2 + 10x_3 - 10x_4 + 4x_5^2 + 2x_6^2 - 2x_7^2 + 5x_{10}^2 + 7x_9 - 11x_1 - 12x_2 + 8x_3 + 2x_4 + 7x_5 + x_6 + 13x_7 + 12x_8 + 9x_9$</td>
</tr>
<tr>
<td></td>
<td><strong>Constraints</strong></td>
</tr>
<tr>
<td></td>
<td>$-10x_1 + 4x_2 + 17x_3 - 2x_4 = 0$</td>
</tr>
<tr>
<td></td>
<td>$3x_1 - 2x_2 - 5x_3 + 2x_4 = 0$</td>
</tr>
<tr>
<td></td>
<td>$-3x_1 - 2x_2 - 5x_3 + 3x_4 + 11x_5 + 2x_6 + 15x_7 + 7x_8 - 3x_9 = 0$</td>
</tr>
<tr>
<td></td>
<td>$0 \leq x_i \leq 10$, $i = 1, \ldots, 10$</td>
</tr>
</tbody>
</table>
4.5. Test problem 5

This problem has eight variables and six inequality constraints (Michalewicz, 1995). It is NLP minimization problem. The optimum solution is 7049.330923. Michalewicz (1995) experienced that this problem is difficult to solve. He has obtained 7377.976. Reported from Deb (2000) the best solution obtained with niching and mutation at maximum generations 4000 and population size 80 has a function value of 7061.221. BASICGA reaches the optimal solution of 7095.4851399999, with the same population size and 10000, which is 0.65% worse than the true optimal objective function value and better then Michalewicz’s one with no addition operators to be applied in it.

4.6. Test problem 6

This problem has seven variables and four non-linear constraints (Michalewicz, 1995). This NLP minimization problem has optimal solution is 680.6308537. Michalewicz (1995) reported the best result of 680.642, which is obtained with penalty function approach in which the penalty parameters are changed with generations. Deb (2000) has found best solution as 680.634 with him constrained handling method. With the combination from roulette-wheel selection for reproduction, arithmetical crossover and non-uniform mutation with 5000 generations we have obtained 680.6591540926304, respectively.

4.7. Test problem 7

This is NLP maximization problem (Himmelblau, 1972). It has three variables and five non-linear constraints. The best reported by Himmelblau (1972) solution is 5280254. BASICGA improves this result. We are obtained 5280335.132 following the combination of genetic operators: square rank selection, blend crossover and non-uniform mutation. The solution is reached at 3000 generations.

---

Table 2 (Continued)

<table>
<thead>
<tr>
<th>No.</th>
<th>Problem description</th>
<th>MIN ( \sum_{j=1}^{n} N_j V_j^p )</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Source of the example—Floudas et al. (1989) and Summanwar et al. (2002)</td>
<td>( \sum_{j=1}^{n} N_j V_j^p )</td>
<td>( H )</td>
</tr>
<tr>
<td></td>
<td>Objective function</td>
<td>( \max \quad 2a_i - 3b_i - 1.5a_i - 2b_i + 0.5a_i )</td>
<td>Constraints</td>
</tr>
</tbody>
</table>
|     |                     | \( V_j \geq S_i, V_j \in J, V_j \in J \) | \( N_j \) \( \leq N_j \), \( N_j \) \( 
\end{align*} 

<p>| 9   | Source of the example—Grossmann and Sargent (1979) | Objective function | Constraints |
|     |                                                   | ( \max \quad 2a_i - 3b_i - 1.5a_i - 2b_i + 0.5a_i ) | ( N_j ) ( \leq N_j ), ( N_j ) ( \in J ) |
|     |                                                   | ( H ) ( \geq H ) | Constraints |
| 10  | Source of the example—Christodoulou (1975) | Objective function | Constraints |
|     |                                                   | ( \max \quad 2a_i - 3b_i - 1.5a_i - 2b_i + 0.5a_i ) | ( N_j ) ( \leq N_j ), ( N_j ) ( \in J ) |</p>
<table>
<thead>
<tr>
<th>No.</th>
<th>Best solutions reported</th>
<th>Optimum obtained by BASIC GA</th>
<th>Values of control variables</th>
<th>BASIC GA settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.97—reported as a best solution found by Deb’s method 4.5705—Summanwar et al. (2002) 4.5705—the best known</td>
<td>4.971699587</td>
<td>0.2 Population number 70</td>
<td>Roulette-wheel selection 1 Blend crossover 0 Generations 1000 1 P-crossover 0.75 P-mutation 0.3 Dynamic penalty function (C_j = 1, B_j = 0.05)</td>
</tr>
<tr>
<td>2</td>
<td>—30664.95—Deb (2000) —30665.41—Summanwar et al. (2002) —30665.54—the best known</td>
<td>−30665.40351175</td>
<td>78 Population number 60</td>
<td>Square rank selection 36.7511842 Blend crossover 5.047273829 Blend crossover 0.919810466 Breeder mutation 1.310946393 Generations 5000 1.398674523 P-crossover 0.9 382.3562487 P-mutation 0.7 Static penalty function (C_j = 5 x 10^-2)</td>
</tr>
<tr>
<td>3</td>
<td>24.34—Deb (2000) 24.36653—Summanwar et al. (2002) 24.3802—the best known</td>
<td>24.4722034</td>
<td>2.218556146 Population number 80</td>
<td>Static penalty function (C_j = 1, B_j = 0.005)</td>
</tr>
<tr>
<td>4</td>
<td>—15—Deb (2000).</td>
<td>−14.999921</td>
<td>1 Population number 150</td>
<td>Static penalty function (C_j = 1, B_j = 0.005)</td>
</tr>
<tr>
<td>5</td>
<td>7060.221—reported as a best solution found by Deb’s method 7049.976—Michalewicz (1995) 7049.330923—the best known solution</td>
<td>7085.4851399599</td>
<td>416.1773613 Population number 80</td>
<td>Roulette-wheel selection 1 Blend crossover 0 Generations 10000 1 P-crossover 0.9 382.3562487 P-mutation 0.7 Static penalty function (C_j = 5 x 10^-2)</td>
</tr>
</tbody>
</table>
Table 3 (Continued)

<table>
<thead>
<tr>
<th>No.</th>
<th>Best solutions reported</th>
<th>Optimum obtained by BASIC GA settings</th>
<th>Values of control variables</th>
<th>BASIC GA settings</th>
</tr>
</thead>
</table>
| 6   | 680.634460—Deb (2000)  
680.6300573—the best known | 680.659116092630 | 2.346 | Populations 90  
1.939 | Roulette-wired selection  
4.39 | Arithmetical crossover  
0.422 | Non-uniform mutation  
1.612 | Generations 5000  
P-crossover 0.9  
P-mutation 0.1  
Static penalty function \(C_j = 4.6 \times 10^5\) |
| 7   | 5280254—Himmelblau (1972) | 5280335.132 | 4.537 | Populations 70  
2.4 | Square rank selection  
60 | Blend crossover  
9.3 | Non-uniform mutation  
7 | Generations 3000  
P-crossover 0.9  
P-mutation 0.1  
Static penalty function \(C_j = 10^5\) |
| 8   | −7.66718—reported as a best solution found by Deb’s method  
−7.667178—Summanwar et al. (2002)  
−7.66—the best known. | −7.6669258098 | 1.117903772 | Populations 70  
1.31031755 | Square rank selection  
1 | Blend crossover  
1 | Non-uniform mutation  
6 | Generations 5000  
P-crossover 0.75  
P-mutation 0.1  
Dynamic penalty function \(C_j = 1, B_j = 0\) |
| 9   | 167427.6571—global optimum found by GAMS | 167484.8218 | 1278.464117 | Populations 180  
1917.690604  
2499.999999  
628.1235683  
310.6410881  
10.00000003  
6.000006001 | Square rank selection  
1 | Blend crossover  
2 | Non-uniform mutation  
2 | Generations 20000  
P-crossover 0.85  
P-mutation 0.2  
Dynamic penalty function \(C_j = 10^5, B_j = 0\) |
| 10  | 15—Christofides (1975) | 15 | −7.667178. The best obtained with Deb’s method solution is −7.6669258098 following the combination of genetic operators: square rank selection, blend crossover and non-uniform mutation. The solution is reached at 3000 generations. |

4.8. Test problem 8

This is MINLP maximization problem taken from Summanwar et al. (2002). It has three binary and two continuous variables and two equality and three inequality constraints. The reported by Summanwar best solution is
4.9. Test problem 9

This is minimization problem taken from Grossmann and Sargent (1979). It has three integer and seven continuous variables and 13 non-linear constraints. The problem is reported as a very hard one for solving. It is involved as a demo problem (bathtubes) in GAMS. Its global optimum obtained by GAMS is 167427.6571. The solution obtained by BASIC GA is 167484.8218. It is obvious that this result is comparable with the global one obtained by GAMS.

4.10. Test problem 10

This is a minimum covering problem taken from Christofides (1975). It has eight binary variables and six inequality constraints. The problem is very easy for solving. The solution obtained by BASIC GA is 15. Except the reported in Table 3 combination of genetic operators, the problem has been tested by application of 1 and 2 point crossover. At this setting the solution very easy traps in a local optimum of value 16 [0, 0, 0, 1, 0, 1, 0, 0]T. To avoid this either number of generations or mutation probability must be increased.

5. Conclusions

A design of the genetic algorithm—called BASIC GA which copes successfully with a range of optimization problems is considered and its architecture described in details.

BASIC GA is created to work with a predetermined constant size of population and to use continuous search space for individuals’ presentation. For the purpose we applied the dynamic real representation scheme for continuous variables and extended it to represent the integer ones. The BASIC GA genetic operators comprise a great number of selection, recombination and mutation schemes. All of them are described in details and their affect on the algorithm convergence and preservation of the populations’ diversity is discussed. Moreover, we have proposed own scheme for selection for replacement, which introduce additional affect on the diversity preservation. BASIC GA provides opportunity to be adjusted to the concrete problems by fitting of its global and local parameters. General guidelines for their setting and for the combination of genetic schemes are given.

The tool is designed to deal with constrained optimization problems by using penalization both, static and dynamic. It is tested on the number of problems. Ten of them, NLP, MINLP and BP are presented as an illustration. The results obtained are comparable with thus reported in the literature.

Finally, BASIC GA provides opportunity for future extension of its capability by involving of new schemes in its genetic operators.

Acknowledgment

The study has been conducted with the financial support of the Bulgarian National Science Fund under the Contract I 1404/04.

References


