The Significance of the Evaluation Function in Evolutionary Algorithms

Zbigniew Michalewicz

Department of Computer Science, University of North Carolina,
Charlotte, NC 28223, USA, *and*
Institute of Computer Science, Polish Academy of Sciences,
ul. Ordona 21, 01-237 Warsaw, Poland

Abstract. The major component of any evolutionary algorithm is its evaluation function, which serves as a major link between the algorithm and the problem being solved. The evaluation function is used to distinguish between better and worse individuals in the population, hence it provides an important feedback for the search process. In this paper we survey a few typical methods for constructing an evaluation function for constrained optimization problems.

Key words: Constrained optimization, evolutionary algorithms, evaluation function, infeasible individuals.

1 Introduction

It is generally accepted that any evolutionary algorithm to solve a problem must have five basic components (Davis, 1987):

- a genetic representation of solutions to the problem,
- a way to create an initial population of solutions,
- an evaluation function (i.e., the environment), rating solutions in terms of their 'fitness',
- 'genetic' operators that alter the genetic composition of children during reproduction, and
- values for the parameters (population size, probabilities of applying genetic operators, etc.)

Most successful implementations of an evolutionary technique for a particular real-world problem require some additional heuristics (problem-specific knowledge) which are incorporated into the basic components listed above. These heuristics apply to genetic representation of solutions, to 'genetic' operators that alter their composition, to values of various parameters, and to methods for creating an initial population. The evaluation function (as the only item out of the above list of five basic components of evolutionary algorithm) usually is taken "for granted"; most researchers did not consider any modifications of evaluation function, since it is often implicitly defined by the problem. However, for

most real-world problems (e.g., constrained problems), the process of selection of an evaluation function might be quite complex by itself, especially when we deal with feasible and infeasible solutions to the problem; several heuristics usually are incorporated in this process. In this paper we examine some of these heuristics and discuss their merits and drawbacks.

The paper is organized as follows. Section 2 states the problem by defining feasible and infeasible individuals and Section 3 provides a discussion on evaluation methods for evolutionary techniques. Section 4 concludes the paper.

2 Feasible and infeasible solutions

The evaluation function serves as the major link between the problem and the evolutionary algorithm. The evaluation function rates individuals in the population: better individuals have better chances for survival and reproduction. Hence it is essential to define an evaluation function which characterizes the problem in a "perfect way". For constrained optimization problems, the issue of handling feasible and infeasible individuals should be addressed very carefully: very often a population contains infeasible individuals but a *feasible* optimal is required. Finding proper evaluation measures for feasible and infeasible individuals is of great importance; it directly influences the outcome (success or failure) of the algorithm.

The issue of processing infeasible individuals is very important for solving constrained optimization problems using evolutionary techniques. For example, in continuous domains, the general nonlinear programming problem[1] is to find $\overline{X}$ so as to

$$\text{optimize } f(\overline{X}),\ \overline{X} = (x_1, \ldots, x_n) \in R^n,$$

where $\overline{X} \in \mathcal{F} \subseteq \mathcal{S}$. The set $\mathcal{S} \subseteq R^n$ defines the search space and the set $\mathcal{F} \subseteq \mathcal{S}$ defines a *feasible* search space. Usually, the search space $\mathcal{S}$ is defined as a n-dimensional rectangle in R^n (domains of variables defined by their lower and upper bounds):

$$l(i) \leq x_i \leq u(i), \quad 1 \leq i \leq n,$$

whereas the feasible set $\mathcal{F}$ is defined by an intersection of $\mathcal{S}$ and a set of additional $m \geq 0$ constraints:

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

Most research on applications of evolutionary computation techniques to nonlinear programming problems was concerned with complex objective functions with $\mathcal{F} = \mathcal{S}$. Several test functions used by various researchers during the last 20 years consider only domains of n variables; this was the case with five test functions F1–F5 proposed by De Jong (1975), as well as with many other test cases proposed since then.

[1] We consider here only continuous variables.

In discrete domains the problem of constraints was acknowledged much earlier. The knapsack problem, the set covering problem, and all types of scheduling and timetabling problems are constrained. Several heuristic methods emerged to handle constraints; however, these methods have not been studied in a systematic way.

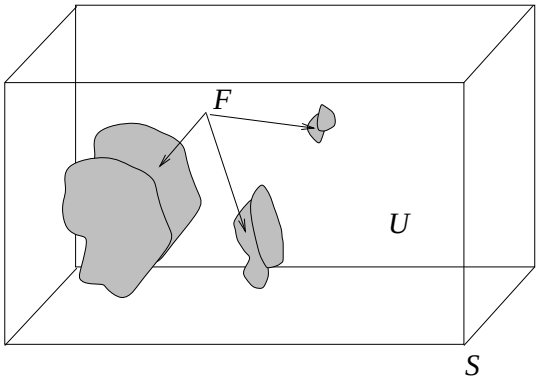

Fig. 1. A search space and its feasible part

In general, a search space $\mathcal{S}$ consists of two disjoint subsets of feasible and infeasible subspaces, $\mathcal{F}$ and $\mathcal{U}$, respectively (see Figure 1). We do not make any assumptions about these subspaces; in particular, they need not be convex and they need not be connected (e.g., as is the case in the example in Figure 1 where the feasible part $\mathcal{F}$ of the search space consists of three disjoined subsets). The fundamental question (which must be addressed by any practitioner) is

> Should we consider infeasible individuals harmful and eliminate them from the population?

The answer 'yes' implies a "death penalty" heuristic, which is a popular option in many evolutionary techniques. Note that rejection of infeasible individuals offers a significant simplification of the algorithm: there is no further need to consider infeasible individuals! There is no need to evaluate them, penalize them, repair them; there is no need to compare them with feasible ones.

The method of eliminating infeasible solutions from a population may work reasonably well when the feasible search space is convex and it constitutes a reasonable part of the whole search space (e.g., evolution strategies do not allow equality constraints since with such constraints the ratio between the sizes of feasible and infeasible search spaces is zero). Otherwise such an approach has serious limitations. For example, for many search problems where the initial population consists of infeasible individuals only, it might be essential to improve them (as opposed to rejecting them). Moreover, quite often the system can reach the optimum solution easier if it is possible to "cross" an infeasible region (especially

in non-convex feasible search spaces). On the top of that, for many constrained problems, the optimum solution lies on the boundary of feasible and infeasible parts of the search space (i.e., the boundary between $\mathcal{F}$ and $\mathcal{U}$); because of that infeasible individuals allow the system to approach the optimum from various directions (as opposed to the case, when a "narrow" feasible corridor leads to the optimum solution). For all these reasons, in most real-world applications, it is essential to process infeasible individuals, so the answer for the question posed in the previous paragraph is simply 'no'!

Consequently, during the search process we have to deal with various feasible and infeasible individuals. The presence of feasible and infeasible individuals in the population influences other parts of the evolutionary algorithm; for example, should the elitist selection method consider a possibility of preserving the best feasible individual, or just the best individual overall? Further, some operators might be applicable to feasible individuals only. However, the major aspect of such a scenario is the need for evaluation of feasible and infeasible individuals. The problem of how to evaluate individuals in the population is far from trivial. In general, we have to design two evaluation functions, $eval_f$ and $eval_i$, for feasible and infeasible domains, respectively.

Several trends for handling infeasible solutions have emerged in the area of evolutionary computation. We discuss them in the following section using examples from discrete and continuous domains.

3 Heuristics for evaluating individuals

In this section we discuss several methods for handling feasible and infeasible solutions in a population; most of these methods emerged quite recently. Only a few years ago Richardson et al. (1989) claimed: "Attempts to apply GA's with constrained optimization problems follow two different paradigms (1) modification of the genetic operators; and (2) penalizing strings which fail to satisfy all the constraints." This is no longer the case as a variety of heuristics have been proposed. Even the category of penalty functions consists of several methods which differ in many important details on how the penalty function is designed and applied to infeasible solutions. Other methods maintain the feasibility of the individuals in the population by means of specialized operators or decoders, impose a restriction that any feasible solution is 'better' than any infeasible solution, consider constraints one at the time in a particular linear order, repair infeasible solutions, use multiobjective optimization techniques, are based on cultural algorithms, or rate solutions using a particular co-evolutionary model.

Before we discuss several constraint-handling techniques, it is worthwhile to point out that (for some problems) even the construction of the evaluation function $eval_f$ for feasible solutions might be far from trivial. For example, for many design problems there are no clear formulae for comparing two feasible designs. Moreover, some problems require optimization of several (possibly conflicting) goals (multi-objective optimization cases), and often problem-dependent heuristics are necessary in such cases to provide with a numerical measure $eval_f(x)$ of

a feasible individual x.

One example to illustrate the problem of evaluating feasible individuals is the satisfiability (SAT) problem. For a given conjunctive normal form formula, say

$$F(x) = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee x_3),$$

it is hard to compare two feasible individuals $p = (0,0,0)$ and $q = (1,0,0)$ (in both cases $F(p) = F(q) = 0$). De Jong and Spears (1989) examined a few options. For example, it is possible to define $eval_i$ to be a ratio of the number of conjuncts which evaluate to true; in that case

$$eval_f(p) = 0.666 \text{ and } eval_f(q) = 0.333.$$

It is also possible (Pardalos 1994) to change the Boolean variables x_i into floating point numbers y_i and to assign, for example,

$$eval'_f(y) = (y_1-1)^2(y_2+1)^2(y_3-1)^2+(y_1+1)^2(y_3+1)^2+(y_2-1)^2(y_3-1)^2.$$

In the above case the solution to the SAT problem corresponds to a set of global minimum points of the objective function: the *true* value of $F(x)$ is equivalent to the global minimum value 0 of $eval_i(y)$.

It may seem, though, that for most optimization problems the evaluation function $eval_f$ for feasible solutions is given. This is the case for numerical optimization problems and for most operation research problems (knapsack problems, traveling salesman problems, set covering problems, etc.) However, it need not be always the case. For example, Falkenauer (1994) rejected the idea of constructing a straightforward $eval_f$ for the bin packing problem (BBP):

> "Let's us define a suitable cost function for the BPP. The objective being to find the minimum number of bins required, the first cost function that comes to mind is simply the number of bins used to 'pack' all the objects. This is correct from a strictly mathematical point of view, but it is unusable in practice. Indeed, such a cost function leads to an extremely unfriendly landscape of the search space: a very small number of optimal points in the space are lost in an exponential number of points where this purported cost function is just one unit above the optimum. Worse, those slightly suboptimal points yield the same cost. The trouble is that such a cost function lacks any capacity of guiding an algorithm in the search, making the problem a 'needle in a haystack'.
> We thus settled for the following cost function for the BPP [...]: maximize
>
> $$f_{BPP} = \frac{\sum_{i=1}^{N}(F_i/C)^k}{N},$$
>
> with N being the number of bins actually used in the solution being evaluated, F_i the sum of sizes of the objects in (the fill of) the bin i, C is the bin capacity, k a constant, $k > 1$.

The constant k expresses our concentration on the 'extremist' bins in comparison to the less filled ones. The larger k is, the more we prefer well-filled 'elite' groups as opposed to a collection of about equally filled bins. In fact, the value of k gives us the possibility to vary the 'ruggedness' of the function to optimize, from the 'needle in a haystack' ($k = 1$, $f_{BPP} = 1/N$) up to the 'best-filled bin' ($k \to \infty$, $f_{BPP} \to \max_i[(F_i/C)^k]$)."

Clearly, the problem of selecting a "perfect" $eval_f$ is far from trivial.

The main three approaches for constructing evaluation function $eval_i$ for infeasible individuals are based on

- penalty functions,
- repair algorithms, and
- special data structures and operators.

We discuss these in turn in the following subsections.

3.1 Penalty functions

It is possible to extend the domain of function $eval_f$ to handle infeasible individuals in the following way:

$$eval_i(x) = eval_f(x) \pm Q(x),$$

where $Q(x)$ represents a penalty for infeasible individual x. The major question is, how should such a penalty function $Q(x)$ be designed? The intuition is simple: the penalty should be kept as low as possible, just above the limit below which infeasible solutions are optimal (the so-called *minimal penalty rule*, see Le Riche et al. 1995). However, it is difficult to implement this rule effectively.

The relationship between infeasible individual 'x' and the feasible part $\mathcal{F}$ of the search space $\mathcal{S}$ plays a significant role in penalizing such individuals: an individual might be penalized just for being infeasible, the 'amount' of its infeasibility is measured to determine the penalty value, or the effort of 'repairing' the individual might be taken into account. However, in such cases a penalty function should consider the "easiness of repairing" an individual as well as the quality of its repaired version; designing such penalty functions is problem-dependent and, in general, quite hard.

Several researchers studied heuristics on design of penalty functions. Some hypotheses were formulated (Richardson et al. 1989):

- "penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints,
- for a problem having few constraints, and few full solutions, penalties which are solely functions of the number of violated constraints are not likely to find solutions,

- good penalty functions can be constructed from two quantities, the *maximum completion cost* and the *expected completion cost*,
- penalties should be close to the *expected completion cost*, but should not frequently fall below it. The more accurate the penalty, the better will be the solutions found. When penalty often underestimates the completion cost, then the search may not find a solution."

and (Siedlecki and Sklanski 1989):

- "the genetic algorithm with a variable penalty coefficient outperforms the fixed penalty factor algorithm,"

where a variability of penalty coefficient was determined by a heuristic rule.

This last observation was further investigated by Smith and Tate (1993). In their work they experimented with dynamic penalties, where the penalty measure depends on the number of violated constraints, the best feasible objective function found, and the best objective function value found.

For numerical optimization problems penalties usually incorporate degrees of constraint violations. Most of these methods use constraint violation measures f_j (for the j-th constraint) for the construction of the $eval_i$; these functions are defined as

$$f_j(\overline{X}) = \begin{cases} \max\{0, g_j(\overline{X})\}, & if\ 1 \leq j \leq q \\ |h_j(\overline{X})|, & if\ q+1 \leq j \leq m \end{cases}$$

For example, Homaifar et al. (1994) assume that for every constraint we establish a family of intervals that determines appropriate penalty values. The method works as follows:

- for each constraint, create several (ℓ) levels of violation,
- for each level of violation and for each constraint, create a penalty coefficient R_{ij} ($i = 1, 2, \ldots, \ell$, $j = 1, 2, \ldots, m$); higher levels of violation require larger values of this coefficient.
- start with a random population of individuals (i.e., these individuals are feasible or infeasible),
- evaluate individuals using the following formula
  $eval(\overline{X}) = f(\overline{X}) + \sum_{j=1}^{m} R_{ij} f_j^2(\overline{X})$,
  where R_{ij} is a penalty coefficient for the i-th level of violation and the j-th constraint.

Note that the function $eval$ is defined on $\mathcal{S}$, i.e., it serves both feasible and infeasible solutions.

It is also possible (as suggested in Siedlecki and Sklanski 1989) to adjust penalties in a dynamic way, taking into account the current state of the search or the generation number. For example, Joines and Houck (1994) assumed dynamic penalties; individuals are evaluated (at the iteration t) by the following formula:

$$eval(\overline{X}) = f(\overline{X}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\overline{X}),$$

where C, α and β are constants. As in Homaifar et al. (1994), the function $eval$ evaluates both feasible and infeasible solutions.

The method is quite similar to Homaifar et al. (1994), but it requires many fewer parameters (C, α and β), and this is independent of the total number of constraints. Also, the penalty component is not constant but changes with the generation number. Instead of defining several levels of violation, the pressure on infeasible solutions is increased due to the $(C \times t)^{\alpha}$ component of the penalty term: towards the end of the process (for high values of t), this component assumes large values.

Michalewicz and Attia (1994) considered the following method:

- divide all constraints into four subsets: linear equations, linear inequalities, nonlinear equations, and nonlinear inequalities,
- select a random single point as a starting point (the initial population consists of copies of this single individual). This initial point satisfies all linear constraints,
- create a set of active constraints A; include there all nonlinear equations and all violated nonlinear inequalities.
- set the initial temperature $\tau = \tau_0$,
- evolve the population using the following formula:
  $eval(\overline{X}, \tau) = f(\overline{X}) + \frac{1}{2\tau} \sum_{j \in A} f_j^2(\overline{X})$,
  (only active constraints are considered),
- if $\tau < \tau_f$, stop, otherwise
  - decrease temperature τ,
  - the best solution serves as a starting point of the next iteration,
  - update the set of active constraints A,
  - repeat the previous step of the main part.

Note that the algorithm maintains the feasibility of all linear constraints using a set of closed operators (see Michalewicz and Janikow (1991)). At every iteration the algorithm considers active constraints only, and so the pressure on infeasible solutions is increased due to the decreasing values of temperature τ. The method requires "starting" and "freezing" temperatures, τ_0 and τ_f, respectively, and the cooling scheme to decrease temperature τ.

A method of adapting penalties was developed by Bean and Hadj-Alouane (1992). As in the previous method, it uses a penalty function, however, one component of the penalty function takes feedback from the search process. Each individual is evaluated by the formula:

$$eval(\overline{X}) = f(\overline{X}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\overline{X}),$$

where $\lambda(t)$ is updated every generation t in the following way:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & if\ \overline{B}(i) \in \mathcal{F}\ for\ all\ t-k+1 \le i \le t \\ \beta_2 \cdot \lambda(t), & if\ \overline{B}(i) \in \mathcal{S} - \mathcal{F}\ for\ all\ t-k+1 \le i \le t \\ \lambda(t), & otherwise, \end{cases}$$

where $\overline{B}(i)$ denotes the best individual, in terms of function $eval$, in generation i, $\beta_1, \beta_2 > 1$ and $\beta_1 \neq \beta_2$ (to avoid cycling). In other words, the method (1) decreases the penalty component $\lambda(t+1)$ for the generation $t+1$, if all best individuals in the last k generations were feasible, and (2) increases penalties, if all best individuals in the last k generations were unfeasible. If there are some feasible and unfeasible individuals as best individuals in the last k generations, $\lambda(t+1)$ remains without change.

Some researchers (Powell and Skolnick 1993, Michalewicz and Xiao 1995) reported good results of their evolutionary algorithms, which worked under the assumption that any feasible individual was better than any infeasible one. Powell and Skolnick (1993) applied this heuristic rule for the numerical optimization problems: evaluations of feasible solutions were mapped into the interval $(-\infty, 1)$ and infeasible solutions—into the interval $(1, \infty)$ (for minimization problems). Michalewicz and Xiao (1995) experimented with the path planning problem and used two separate evaluation functions for feasible and infeasible individuals. The values of $eval_i$ were increased (i.e., made less attractive) by adding such a constant, so that the best infeasible individual was worse that the worst feasible one. This approach can be viewed as 'adaptive' penalty, which is a function of the values of individuals in the current population.

Yet another approach was proposed recently by Le Riche et al. 1995. The authors designed a (segregated) genetic algorithm which uses two values of penalty parameters (for each constraint) instead of one; these two values aim at achieving a balance between heavy and moderate penalties by maintaining two subpopulations of individuals. The population is split into two cooperating groups, where individuals in each group are evaluated using either one of the two penalty parameters.

It seems that the appropriate choice of the penalty method may depend on (1) the ratio between sizes of the feasible and the whole search space, (2) the topological properties of the feasible search space, (3) the type of the objective function, (4) the number of variables, (5) number of constraints, (6) types of constraints, and (7) number of active constraints at the optimum. Thus the use of penalty functions is not trivial and only some partial analysis of their properties is available. Also, a promising direction for applying penalty functions is the use of adaptive penalties: penalty factors can be incorporated in the chromosome structures in a similar way as some control parameters are represented in the structures of evolution strategies and evolutionary programming.

3.2 Repair methods

Repair algorithms enjoy a particular popularity in the evolutionary computation community: for many combinatorial optimization problems (e.g., traveling salesman problem, knapsack problem, set covering problem, etc.) it is relatively easy to 'repair' an infeasible individual. Such a repaired version can be used either for evaluation only, i.e.,

$$eval_i(y) = eval_f(x),$$

where x is a repaired (i.e., feasible) version of y, or it can also replace (with some probability) the original individual in the population.

The process of repairing infeasible individuals is related to a combination of learning and evolution (the so-called *Baldwin effect*, Whitley et al. 1994). Learning (as local search in general, and local search for the closest feasible solution, in particular) and evolution interact with each other: the fitness value of the improvement is transferred to the individual. In that way a local search is analogous to learning that occurs during one generation of a particular string.

The weakness of these methods is in their problem dependence. For each particular problem a specific repair algorithm should be designed. Moreover, there are no standard heuristics on design of such algorithms: usually it is possible to use a greedy repair, random repair, or any other heuristic which would guide the repair process. Also, for some problems the process of repairing infeasible individuals might be as complex as solving the original problem. This is the case for the nonlinear transportation problem (see Michalewicz 1993), most scheduling and timetable problems, and many others.

On the other hand, the recently completed Genocop III system (Michalewicz and Nazhiyath 1995) for constrained numerical optimization (nonlinear constraints) is based on repair algorithms. Genocop III incorporates the original Genocop system (which handles linear constraints only; see section H), but also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population P_s consists of so-called search points which satisfy linear constraints of the problem. As in Genocop, the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators. The second population P_r consists of so-called reference points from $\mathcal{F}$; these points are fully feasible, i.e., they satisfy *all* constraints. Reference points $\mathbf{r}$ from P_r, being feasible, are evaluated directly by the objective function (i.e., $eval_f(\mathbf{r}) = f(\mathbf{r})$). On the other hand, search points from P_s are "repaired" for evaluation and the repair process works as follows. Assume, there is a search point $\mathbf{s} \in P_s$. If $\mathbf{s} \in \mathcal{F}$, then $eval_f(\mathbf{s}) = f(\mathbf{s})$, since $\mathbf{s}$ is fully feasible. Otherwise (i.e., $\mathbf{s} \notin \mathcal{F}$), the system selects one of the reference points, say $\mathbf{r}$ from P_r and creates a sequence of points $\mathbf{z}$ from a segment between $\mathbf{s}$ and $\mathbf{r}$: $\mathbf{z} = a\mathbf{s} + (1-a)\mathbf{r}$. This can be done either (1) in a random way by generating random numbers a from the range $\langle 0, 1\rangle$, or (2) in a deterministic way by setting $a_i = 1/2, 1/4, 1/8, \ldots$ until a feasible point is found. Once a fully feasible $\mathbf{z}$ is found, $eval_i(\mathbf{s}) = eval_f(\mathbf{z}) = f(\mathbf{z})$. Clearly, in different generations the same search point $\mathcal{S}$ can evaluate to different values due to the random nature of the repair process.

The question of replacing repaired individuals is related to so-called *Lamarckian evolution* (Whitley et al. 1994), which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the chromosome. As stated in Whitley et al. 1994:

> "Our analytical and empirical results indicate that Lamarckian strategies are often an extremely fast form of search. However, functions exist where both the simple genetic algorithm without learning and the

> Lamarckian strategy used [...] converge to local optima while the simple genetic algorithm exploiting the Baldwin effect converges to a global optimum."

This is why it is necessary to use the replacement strategy very carefully.

Recently (see Orvosh and Davis 1993) a so-called 5%-rule was reported: this heuristic rule states that in many combinatorial optimization problems, an evolutionary computation technique with a repair algorithm provides the best results when 5% of repaired individuals replace their infeasible originals. However, many recent experiments (e.g., Michalewicz 1994) indicated that for many combinatorial optimization problems this rule did not apply. Either a different percentage gives better results, or there is no significant difference in the performance of the algorithm for various probabilities of replacement.

In continuous domains, a new replacement rule is emerging. The Genocop III system (see section E) for constrained numerical optimization problems with nonlinear constraints is based on the repair approach. The first experiments (based on 10 test cases which have various numbers of variables, constraints, types of constraints, numbers of active constraints at the optimum, etc.) indicate that the 15% replacement rule is a clear winner: the results of the system are much better than with either lower or higher values of the replacement rate.

At present, it seems that the 'optimal' probability of replacement is problem-dependent and it may change over the evolution process as well. Further research is required for comparing different heuristics for setting this parameter, which is of great importance for all repair-based methods.

3.3 Specialized data structures and operators

There are some other possibilities for handling constraints by evolutionary algorithms. One of the popular directions is based on maintenance of feasible populations by special representations and genetic operators.

During the last decade several specialized systems were developed for particular optimization problems; these systems use a unique chromosomal representations and specialized 'genetic' operators which alter their composition. Some of such systems were described in Davis (1991); other examples include various systems developed for the traveling salesman problem (Michalewicz, 1996), as well as Genocop (Michalewicz and Janikow 1991) for optimizing numerical functions with linear constraints and Genetic-2N (Michalewicz et al. 1991) for the nonlinear transportation problem. For example, Genocop assumes linear constraints only and a feasible starting point (or feasible initial population). A closed set of operators maintains feasibility of solutions. For example, when a particular component x_i of a solution vector $\overline{X}$ is mutated, the system determines its current domain $dom(x_i)$ (which is a function of linear constraints and remaining values of the solution vector $\overline{X}$) and the new value of x_i is taken from this domain (either with flat probability distribution for uniform mutation, or other probability distributions for non-uniform and boundary mutations). In any case the

offspring solution vector is always feasible. Similarly, arithmetic crossover[2]

$$a\overline{X} + (1-a)\overline{Y}$$

of two feasible solution vectors $\overline{X}$ and $\overline{Y}$ yields always a feasible solution (for $0 \leq a \leq 1$) in convex search spaces (the system assumes linear constraints only which imply convexity of the feasible search space $\mathcal{F}$). Consequently, there is no need to define the function $eval_i$; the function $eval_f$ is (as usual) the objective function f.

Such systems are much more reliable than any other evolutionary techniques based on the penalty approach (Michalewicz 1994). This is a quite popular trend. Many practitioners use problem-specific representations and specialized operators in building very successful evolutionary algorithms in many areas; these include numerical optimization, machine learning, optimal control, cognitive modeling, classic operation research problems (traveling salesman problem, knapsack problems, transportation problems, assignment problems, bin packing, scheduling, partitioning, etc.), engineering design, system integration, iterated games, robotics, signal processing, and many others.

Also, it is interesting to note, that original evolutionary programming techniques (Fogel et al. 1966) and genetic programming techniques (Koza 1992) fall into this category of evolutionary algorithms: these techniques maintain feasibility of finite state machines or hierarchically structured programs by means of specialized representations and operators.

Another possibility for restricting the search to feasible individuals only is based on the idea of decoders. In these techniques a chromosome "gives instructions" on how to build a feasible solution. For example, a sequence of items for the knapsack problem can be interpreted as: "take an item if possible"—such interpretation would lead always to feasible solutions. Let us consider the following scenario: we try to solve the 0–1 knapsack problem with n items; the profit and weight of the i-th item are p_i and w_i, respectively. We can sort all items in decreasing order of p_i/w_i's and interpret the binary string

(1100110001001110101001010111010101...0010)

in the following way: take the first item from the list (i.e., the item with the largest ration of profit to weight) if the item fits in the knapsack. Continue with the second, fifth, sixth, tenth, etc. items from the sorted list, until the knapsack is full or there are no more items available. Note that the sequence of all 1's corresponds to a greedy solution. Any sequence of bits would translate into a feasible solution, every feasible solution may have many possible codes. We can apply classical binary operators (crossover and mutation): any offspring is clearly feasible.

[2] The arithmetical crossover operator generate offspring by linear combinations of the parents. As noted, such a strategy of generating a set of diverse trial points by linear and convex combinations (and allowing the offspring to influence the search) was proposed some years ago in the scatter search approach by Glover (1977).

However, it is important to point out that several factors should be taken into account while using decoders. Each decoder imposes a relationship T between a feasible solution and decoded solution. It is important that several conditions are satisfied: (1) for each solution $s \in \mathcal{F}$ there is a decoded solution d, (2) each decoded solution d corresponds to a feasible solution s, and (3) all solutions in $\mathcal{F}$ should be represented by the same number of decodings d.[3]

Additionally, it is reasonable to request that (4) the transformation T is computationally fast and (5) it has locality feature in the sense that small changes in the decoded solution result in small changes in the solution itself. An interesting study on coding trees in genetic algorithm was reported by Palmer and Kershenbaum (1994), where the above conditions were formulated.

The third option for maintaining feasible individuals only is based on the idea of exploring boundaries between feasible and infeasible parts of the search space. Some other heuristic methods recognized the need for searching areas close to the boundary of the feasible region. For example, one of the most recently developed approaches for constrained optimization is strategic oscillation. Strategic oscillation was originally proposed in accompaniment with the strategy of scatter search (Glover, 1977), and more recently has been applied to a variety of problem settings in combinatorial and nonlinear optimization (see, for example, the review of Glover, 1995). The approach is based on identifying a critical level, which represents a boundary between feasibility and infeasibility. The basic strategy is to approach and cross the feasibility boundary, by a design that is implemented either by adaptive penalties and inducements (which are progressively relaxed or tightened according to whether the current direction of search is to move deeper into a particular region or to move back toward the boundary) or by simply employing modified gradients or sub-gradients to progress in the desired direction.

It seems that the evolutionary computation techniques have a huge potential in incorporating specialized operators which search the boundary of feasible and infeasible regions in an efficient way. The first results were reported in Michalewicz et al. (1996); for resent results, see Schoenauer and Michalewicz (1996, 1997).

[3] However, as observed by Davis (1997), the requirement that all solutions in $\mathcal{F}$ should be represented by the same number of decodings seems overly strong: there are cases in which this requirement might be suboptimal. For example, suppose we have a decoding and encoding procedure which makes it impossible to represent suboptimal solutions, and which encodes the optimal one: this might be a good thing. (An example would be a graph coloring order-based chromosome, with a decoding procedure that gives each node its first legal color. This representation could not encode solutions where some nodes that could be colored were not colored, but this is a good thing!)

4 Conclusions

This paper surveys several methods which support the most important step of any evolutionary technique: evaluation of the population. It is clear that further studies in this area are necessary: different problems require different "treatment". It is also possible to mix different strategies described in this paper; for example, Paechter et al. 1994 built a successful evolutionary system for a timetable problem, where "each chromosome in the population gives instructions on how to build a timetable. These instruction may or may not result in a feasible timetable", thus allowing other heuristics to be added to the proposed decoder. The author is not aware of any results which provide heuristics on relationships between categories of optimization problems and evaluation techniques in the presence of infeasible individuals; this is an important area of future research.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant IRI-9322400.

References

Bean, J.C. and Hadj-Alouane, A.B. (1992). A Dual Genetic Algorithm for Bounded Integer Programs. Department of Industrial and Operations Engineering, The University of Michigan, TR 92-53.

Davis, L. (1987). *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, 1987.

Davis, L. (1991). *Handbook of Genetic Algorithms*, New York, Van Nostrand Reinhold.

Davis, L. (1997). Private communication.

De Jong, K.A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Doctoral dissertation, University of Michigan, *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No 76-9381).

De Jong K.A. and W.M. Spears (1989). Using Genetic Algorithms to Solve NP-Complete Problems. In *Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 124–132.

Falkenauer, E. (1994). A New Representation and Operators for GAs Applied to Grouping Problems. Evolutionary Computation, Vol.2, No.2, pp.123–144.

Fogel, L.J., A.J. Owens and M.J. Walsh (1966). *Artificial Intelligence through Simulated Evolution*, New York, Wiley.

Glover, F. (1977). Heuristics for Integer Programming Using Surrogate Constraints. Decision Sciences, Vol.8, No.1, 156–166.

Glover, F. (1995). Tabu Search Fundamentals and Uses. Graduate School of Business, University of Colorado.

Homaifar, A., S. H.-Y. Lai and X. Qi (1994). Constrained Optimization via Genetic Algorithms. Simulation, Vol.62, 242–254.

Joines, J.A. and C.R. Houck (1994). On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems With GAs. In *Proceedings of the Evolutionary Computation Conference—Poster Sessions*, part of the IEEE World Congress on Computational Intelligence, Orlando, 27–29 June 1994, 579–584.

Koza, J.R. (1992). *Genetic Programming*, Cambridge, MA, MIT Press.

Le Riche, R., C. Vayssade, R.T. Haftka (1995). A Segregated Genetic Algorithm for Constrained Optimization in Structural Mechanics. Technical Report, Universite de Technologie de Compiegne, France.

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

Michalewicz, Z. and N. Attia (1994). In Evolutionary Optimization of Constrained Problems. *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, eds. A.V. Sebald and L.J. Fogel, River Edge, NJ, World Scientific Publishing, 98–108.

Michalewicz, Z. and C. Janikow (1991). Handling Constraints in Genetic Algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 151–157.

Michalewicz, Z. and Nazhiyath, G. (1995). Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In *Proceedings of the 2nd IEEE International Conference on Evolutionary Computation*, Perth, 29 November – 1 December 1995.

Michalewicz, Z., Nazhiyath, G., and Michalewicz, M. (1996). A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems. In *Proceedings of the 5th Annual Conference on Evolutionary Programming*, L.J. Fogel, P.J. Angeline, and T. Baeck (eds.), MIT Press, Cambridge, MA, 1996, pp.305–312.

Michalewicz, Z., G.A. Vignaux, and M. Hobbs (1991). A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem. ORSA Journal on Computing, Vol.3, No.4, 1991, 307–316.

Michalewicz, Z. and J. Xiao (1995). Evaluation of Paths in Evolutionary Planner/ Navigator. In *Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems*, Tokyo, Japan, May 30–31, 1995, pp.45–52.

Orvosh, D. and L. Davis (1993). Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 650.

Paechter, B., A. Cumming, H. Luchian, and M. Petriuc (1994). Two Solutions to the General Timetable Problem Using Evolutionary Methods. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 27–29 June 1994, 300–305.

Palmer, C.C. and A. Kershenbaum (1994). Representing Trees in Genetic Algorithms. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 27–29 June 1994, 379–384.

Pardalos, P. (1994). On the Passage from Local to Global in Optimization. In *Mathematical Programming*, J.R. Birge and K.G. Murty (Editors), The University of Michigan, 1994.

Powell, D. and M.M. Skolnick (1993). Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 424–430.

Richardson, J.T., M.R. Palmer, G. Liepins and M. Hilliard (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 191–197.

Schoenauer, M. and Michalewicz, Z. (1996). Evolutionary Computation at the Edge of Feasibility. In *Proceedings of the 4th Parallel Problem Solving from Nature*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (Editors), Berlin, September 22–27, 1996, Springer-Verlag, Lecture Notes in Computer Science, Vol.1141, pp.245–254.

Schoenauer, M. and Michalewicz, Z. (1997). Boundary Operators for Constrained Parameter Optimization Problems. In *Proceedings of the 7th International Conference on Genetic Algorithms*, East Lansing, Michigan, July 19–23, 1997.

Siedlecki, W. and J. Sklanski (1989). Constrained Genetic Optimization via Dynamic Reward–Penalty Balancing and Its Use in Pattern Recognition. In *Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 141–150.

Smith, A.E. and D.M. Tate (1993). Genetic Optimization Using a Penalty Function. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, 499–503, Urbana-Champaign, CA: Morgan Kaufmann.

Whitley, D., V.S. Gordon, and K. Mathias (1994). Lamarckian Evolution, the Baldwin Effect and function Optimization. In *Proceedings of the Parallel Problem Solving from Nature, 3*, Springer-Verlag, New York, 6–15.

This article was processed using the LaTeX macro package with LLNCS style