Constraint-Handling Techniques used with Evolutionary Algorithms

Carlos A. Coello Coello

CINVESTAV-IPN

Mexico City, México

ccoello@cs.cinvestav.mx

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan.
 ISBN 978-1-4503-5764-7/18/07
DOI: https://doi.org/10.1145/3205651.3207855

Motivation

Traditional mathematical programming techniques used to solve constrained optimization problems have several limitations when dealing with the general nonlinear programming problem:

$$\text{Min } f(\vec{x}) \tag{1}$$

subject to:

$$g_i(\vec{x}) \leq 0, \quad i = 1, \ldots, m \tag{2}$$

$$h_j(\vec{x}) = 0, j = 1, \ldots, p \tag{3}$$

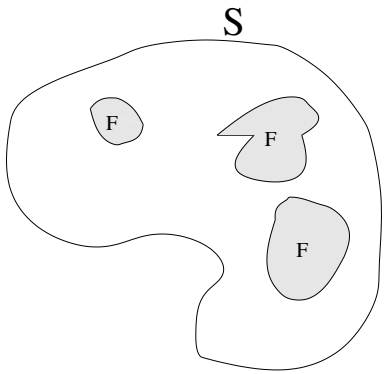
where $\vec{x}$ is the vector of decision variables $\vec{x} = [x_1, x_2, \ldots, x_n]^T$, m is the number of inequality constraints and p is the number of equality constraints (in both cases, constraints can be either linear or nonlinear).

Motivation

Evolutionary Algorithms (EAs) have been found successful in the solution of a wide variety of optimization problems. However, EAs are unconstrained search techniques. Thus, incorporating constraints into the fitness function of an EA is an open research area.

There is a considerable amount of research regarding mechanisms that allow EAs to deal with equality and inequality constraints; both type of constraints can be linear or nonlinear. Such work is precisely the scope of this tutorial.

Search Space



A Taxonomy of Constraint-Handling Approaches

- Penalty Functions
- Special representations and operators
- Repair algorithms
- Separation of constraints and objectives
- Hybrid Methods

Penalty Functions

The most common approach in the EA community to handle constraints (particularly, inequality constraints) is to use penalties. Penalty functions were originally proposed by Richard Courant in the 1940s and were later expanded by Carroll and Fiacco & McCormick.

Penalty Functions

The idea of penalty functions is to transform a constrained optimization problem into an uncontrained one by adding (or subtracting) a certain value to/from the objective function based on the amount of constraint violation present in a certain solution.

Penalty Functions

In mathematical programming, two kinds of penalty functions are considered: exterior and interior. In the case of **exterior** methods, we start with an infeasible solution and from there we move towards the feasible region.

Penalty Functions

In the case of **interior methods**, the penalty term is chosen such that its value will be small at points away from the constraint boundaries and will tend to infinity as the constraint boundaries are approached. Then, if we start from a feasible point, the subsequent points generated will always lie within the feasible region since the constraint boundaries act as barriers during the optimization process.

Penalty Functions

EAs normally adopt external penalty functions of the form:

$$\phi(\vec{x}) = f(\vec{x}) \pm \left[\sum_{i=1}^{n} r_i \times G_i + \sum_{j=1}^{p} c_j \times L_j \right] \tag{4}$$

where $\phi(\vec{x})$ is the new (expanded) objective function to be optimized, G_i and L_j are functions of the constraints $g_i(\vec{x})$ and $h_j(\vec{x})$, respectively, and r_i and c_j are positive constants normally called "penalty factors".

Penalty Functions

The most common form of G_i and L_j is:

$$G_i = \max[0, g_i(\vec{x})]^{\beta} \tag{5}$$

$$L_j = |h_j(\vec{x})|^{\gamma} \tag{6}$$

where β and γ are normally 1 or 2.

Penalty Functions

Penalty functions can deal both with equality and inequality constraints, and the normal approach is to transform an equality to an inequality of the form:

$$|h_j(\vec{x})| - \epsilon \leq 0 \tag{7}$$

where ϵ is the tolerance allowed (a very small value).

Types of Penalty Functions used with EAs

- Death Penalty
- Static Penalty
- Dynamic Penalty
- Adaptive Penalty
- Other Approaches
  - Self-Adaptive Fitness Formulation
  - ASCHEA
  - Stochastic Ranking

Death Penalty

The rejection of infeasible individuals (also called "death penalty") is probably the easiest way to handle constraints and it is also computationally efficient, because when a certain solution violates a constraint, it is rejected and generated again. Thus, no further calculations are necessary to estimate the degree of infeasibility of such a solution.

Criticism to Death Penalty

- Not advisable, except in the case of problems in which the feasible region is fairly large.
- No use of information from infeasible points.
- Search may "stagnate" in the presence of very small feasible regions.
- A variation that assigns a zero fitness to infeasible solutions may work better in practice.

Static Penalty

Under this category, we consider approaches in which the penalty factors do not depend on the current generation number in any way, and therefore, remain constant during the entire evolutionary process.

Static Penalty

An example of this sort of approach is the following:

- The approach proposed by Homaifar, Lai and Qi [1994] in which they define levels of violation of the constraints (and penalty factors associated to them):

$$\text{fitness}(\vec{x}) = f(\vec{x}) + \sum_{i=1}^{m} \left(R_{k,i} \times \ \max\left[0, g_i(\vec{x})\right]^2 \right) \tag{8}$$

  where $R_{k,i}$ are the penalty coefficients used, m is total the number of constraints, $f(\vec{x})$ is the unpenalized objective function, and $k = 1, 2, \ldots, l$, where l is the number of levels of violation defined by the user.

Criticism to Static Penalty

- It may not be a good idea to keep the same penalty factors along the entire evolutionary process.
- Penalty factors are, in general, problem-dependent.
- Approach is simple, although in some cases (e.g., in the approach by Homaifar, Lai and Qi [1994]), the user may need to set up a high number of penalty factors.

Dynamic Penalty

Within this category, we will consider any penalty function in which the current generation number is involved in the computation of the corresponding penalty factors (normally the penalty factors are defined in such a way that they increase over time—i.e., generations).

Dynamic Penalty

An example of this sort of approach is the following:

- The approach from Joines and Houck [1994] in which individuals are evaluated (at generation t) using:

$$\text{fitness}(\vec{x}) = f(\vec{x}) + (C \times t)^{\alpha} \times SVC(\beta, \vec{x}) \tag{9}$$

  where C, α and β are constants defined by the user (the authors used $C = 0.5$, $\alpha = 1$ or 2, and $\beta = 1$ or 2).

Dynamic Penalty

$SVC(\beta, \vec{x})$ is defined as:

$$SVC(\beta, \vec{x}) = \sum_{i=1}^{n} D_i^{\beta}(\vec{x}) + \sum_{j=1}^{p} D_j(\vec{x}) \tag{10}$$

and

$$D_i(\vec{x}) = \begin{cases} 0 & g_i(\vec{x}) \leq 0 \\ |g_i(\vec{x})| & \text{otherwise} \end{cases} \quad 1 \leq i \leq n \tag{11}$$

$$D_j(\vec{x}) = \begin{cases} 0 & -\epsilon \leq h_j(\vec{x}) \leq \epsilon \\ |h_j(\vec{x})| & \text{otherwise} \end{cases} \quad 1 \leq j \leq p \tag{12}$$

This dynamic function increases the penalty as we progress through generations.

Criticism to Dynamic Penalty

- Some researchers have argued that dynamic penalties work better than static penalties.
- In fact, many EC researchers consider dynamic penalty as a good choice when dealing with an arbitrary constrained optimization problem.
- Note however, that it is difficult to derive good dynamic penalty functions in practice as it is difficult to produce good penalty factors for static functions.

Adaptive Penalty

Bean and Hadj-Alouane [1992,1997] developed a method that uses a penalty function which takes a feedback from the search process. Each individual is evaluated by the formula:

$$\text{fitness}(\vec{x}) = f(\vec{x}) + \lambda(t) \left[\sum_{i=1}^{n} g_i^2(\vec{x}) + \sum_{j=1}^{p} |h_j(\vec{x})| \right] \tag{13}$$

where $\lambda(t)$ is updated at every generation t.

Adaptive Penalty

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

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & \text{if case \#1} \\ \beta_2 \cdot \lambda(t), & \text{if case \#2} \\ \lambda(t), & \text{otherwise,} \end{cases} \tag{14}$$

where cases #1 and #2 denote situations where the best individual in the last k generations was always (case #1) or was never (case #2) feasible, $\beta_1, \beta_2 > 1$, $\beta_1 > \beta_2$, and $\beta_1 \neq \beta_2$ (to avoid cycling).

Adaptive Penalty

In other words, the penalty component $\lambda(t+1)$ for the generation $t+1$ is decreased if all the best individuals in the last k generations were feasible or is increased if they were all infeasible. If there are some feasible and infeasible individuals tied as best in the population, then the penalty does not change.

Criticism to Adaptive Penalty

- Setting the parameters of this type of approach may be difficult (e.g., what generational gap (k) is appropriate?).
- This sort of approach regulates in a more "intelligent" way the penalty factors.
- An interesting aspect of this approach is that it tries to avoid having either an all-feasible or an all-infeasible population. Other constraint-handling approaches pay a lot of attention to this issue.

Penalty Functions: Central Issues

The main problem with penalty functions is that the "ideal" penalty factor to be adopted in a penalty function cannot be known *a priori* for an arbitrary problem. If the penalty adopted is too high or too low, then there can be problems.

Penalty Functions: Central Issues

If the penalty is too high and the optimum lies at the boundary of the feasible region, the EA will be pushed inside the feasible region very quickly, and will not be able to move back towards the boundary with the infeasible region. On the other hand, if the penalty is too low, a lot of the search time will be spent exploring the infeasible region because the penalty will be negligible with respect to the objective function.

Other Approaches

Three modern constraint-handling approaches that use penalty functions deserve special consideration, since they are highly competitive:

- Self-Adaptive Fitness Formulation
- ASCHEA
- Stochastic Ranking
- The α Constrained Method

Self-Adaptive Fitness Formulation

- Proposed by Farmani and Wright [2003].
- The approach uses an adaptive penalty that is applied in 3 steps:
  1. The sum of constraint violation is computed for each individual.
  2. The best and worst solutions are identified in the current population.
  3. A penalty is applied in two parts:
     - It is applied only if one or more feasible solutions have a better objective function value than the best solution found so far. The idea is to increase the fitness of the infeasible solutions.
     - Increase the fitness of the infeasible solutions as to favor those solutions which are nearly feasible and also have a good objective function value.

Self-Adaptive Fitness Formulation

- The penalty factor is defined in terms of both the best and the worst solutions.
- The authors use a genetic algorithm with binary representation (with Gray codes) and roulette-wheel selection.
- Good results, but not better than the state-of-the-art techniques (e.g., Stochastic Ranking).

Self-Adaptive Fitness Formulation

- The number of fitness function evaluations required by the approach is high $(1,400,000)$.
- Its main selling point is that the approach does not require of any extra user-defined parameters. Also, the implementation seems relatively simple.
- Other self-adaptive penalty functions have also been proposed (see for example [Tessema & Yen, 2006]).

ASCHEA

The Adaptive Segregational Constraint Handling Evolutionary Algorithm (ASCHEA) was proposed by Hamida and Schoenauer [2000]. It uses an evolution strategy and it is based on three main components:

- An adaptive penalty function.
- A recombination guided by the constraints.
- A so-called "segregational" selection operator.

ASCHEA

The **adaptive penalty** adopted is the following:

$$fitness(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if the solution is feasible} \\ f(\vec{x}) - penal(\vec{x}) & \text{otherwise} \end{cases} \quad (15)$$

where

$$\text{penal}(\vec{x}) = \alpha \sum_{j=1}^{q} g_j^+(\vec{x}) + \alpha \sum_{j=q+1}^{m} |h_j(\vec{x})| \quad (16)$$

where $g_j^+(\vec{x})$ is the positive part of $g_j(\vec{x})$ and α is the penalty factor adopted for all the constraints.

ASCHEA

The penalty factor is adapted based on the desired ratio of feasible solutions (with respect to the entire population) τ_{target} and the current ratio at generation t τ_t:

$$\begin{aligned} \text{if}(\tau_t > \tau_{target}) \quad & \alpha(t+1) = \alpha(t)/fact \\ \text{else} \quad & \alpha(t+1) = \alpha(t) * fact \end{aligned}$$

where $fact > 1$ and τ_{target} are user-defined parameters and

$$\alpha(0) = \left| \frac{\sum_{i=1}^{n} f_i(\vec{x})}{\sum_{i=1}^{n} V_i(\vec{x})} \right| * 1000 \quad (17)$$

where $V_i(\vec{x})$ is the sum of the constraint violation of individual i.

ASCHEA

The **Recombination guided by the constraints** combines an infeasible solution with a feasible one when there are few feasible solutions, based on τ_{target}. If $\tau_t > \tau_{target}$, then the recombination is performed in the traditional way (i.e., disregarding feasibility).

The **Segregational Selection** operator aims to define a ratio τ_{select} of feasible solutions such that they become part of the next generation. The remaining individuals are selected in the traditional way based on their penalized fitness. τ_{select} is another user-defined parameter.

ASCHEA

- In its most recent version [2002], it uses a penalty factor for each constraint, as to allow more accurate penalties.
- This version also uses niching to maintain diversity (this, however, adds more user-defined parameters).
- The approach requires a high number of fitness function evaluations $(1,500,000)$.

Stochastic Ranking

This approach was proposed by Runarsson and Yao [2000], and it consists of a multimembered evolution strategy that uses a penalty function and a selection based on a ranking process. The idea of the approach is try to balance the influence of the objective function and the penalty function when assigning fitness to an individual. An interesting aspect of the approach is that it doesn't require the definition of a penalty factor. Instead, the approach requires a user-defined parameter called P_f, which determines the balance between the objective function and the penalty function.

Stochastic Ranking

```
Begin
    For i=1 to N
        For j=1 to P-1
            u=random(0,1)
            If (φ(I_j) = φ(I_{j+1}) = 0) or (u < P_f)
                If (f(I_j) > f(I_{j+1}))
                    swap(I_j,I_{j+1})
            Else
                If (φ(I_j) > φ(I_{j+1}))
                    swap(I_j,I_{j+1})
        End For
        If no swap is performed
            break
    End For
End
```

Stochastic Ranking

The population is sorted using an algorithm similar to bubble-sort (which sorts a list based on pairwise comparisons). Based on the value of P_f, the comparison of two adjacent individuals is performed based only on the objective function. The remainder of the comparisons take place based on the sum of constraint violation. Thus, P_f introduces the "stochastic" component to the ranking process, so that some solutions may get a good rank even if they are infeasible.

Stochastic Ranking

- The value of P_f certainly impacts the performance of the approach. The authors empirically found that $0.4 < P_f < 0.5$ produces the best results.
- The authors report the best results found so far for the benchmark adopted with only $350,000$ fitness function evaluations.
- The approach is easy to implement.
- In 2005, an improved version of SR was introduced. In this case, the authors use evolution strategies and differential variation.

Miscellaneous Approaches

There are two other approaches that we will also briefly discuss:

- A Simple Multimembered Evolution Strategy (SMES)
- The α Constrained Method

A Simple Multimembered Evolution Strategy

Mezura-Montes and Coello [2005] proposed an approach based on a $(\mu + \lambda)$ evolution strategy. Individuals are compared using the following criteria (originally proposed by [Deb, 2000]):

1. Between two feasible solutions, the one with the highest fitness value wins.
2. If one solution is feasible and the other one is infeasible, the feasible solution wins.
3. If both solutions are infeasible, the one with the lowest sum of constraint violation is preferred.

A Simple Multimembered Evolution Strategy

Additionally, the approach has 3 main mechanisms:

1. **Diversity Mechanism**: The infeasible solution which is closest to become feasible is retained in the population, so that it is recombined with feasible solutions.
2. **Combined Recombination**: Panmictic recombination is adopted, but with a combination of the discrete and intermediate recombination operators.
3. **Stepsize**: The initial stepsize of the evolution strategy is reduced so that finer movements in the search space are favored.

A Simple Multimembered Evolution Strategy

This approach provided highly competitive results with respect to stochastic ranking, the homomorphous maps and ASCHEA while performing only 240,000 fitness function evaluations.

In a further paper, similar mechanisms were incorporated into a differential evolution algorithm, obtaining even better results.

The approach is easy to implement and robust.

The α Constrained Method

This is a transformation method for constrained optimization introduced by Takahama [1999]. Its main idea is to define a satisfaction level for the constraints of a problem. The approach basically adopts a lexicographic order with relaxation of the constraints. Equality constraints can be easily handled through the relaxation of the constraints.

The α Constrained Method

In [Takahama and Sakai, 2005], this approach is coupled to a modified version of Nelder and Mead's method. The authors argue that Nelder and Mead's method can be seen as an evolutionary algorithm in which, for example, the variation operators are: reflection, contraction and expansion. The authors also extend this method with a boundary mutation operator, the use of multiple simplexes, and a modification to the traditional operators of the method, as to avoid that the method gets easily trapped in a local optimum.

The α Constrained Method

The approach was validated using a well-known benchmark of 13 test functions. Results were compared with respect to stochastic ranking. The number of evaluations performed was variable and ranged from 290,000 to 330,000 evaluations in most cases. The results found were very competitive, although the approach had certain sensitivity to the variation of some of its parameters. In a further paper, Takahama and Sakai [2006] another technique for relaxing the constraints, but using a parameter called ϵ. In this case, differential evolution is the search engine, and a gradient-based mutation operator is adopted.

Special representations and operators

Some researchers have decided to develop special representation schemes to tackle a certain (particularly difficult) problem for which a generic representation scheme (e.g., the binary representation used in the traditional genetic algorithm) might not be appropriate.

Special representations and operators

Due to the change of representation, it is necessary to design special genetic operators that work in a similar way than the traditional operators used with a binary representation. For example: Random Keys [Bean, 1992, 1994].

Special representations and operators

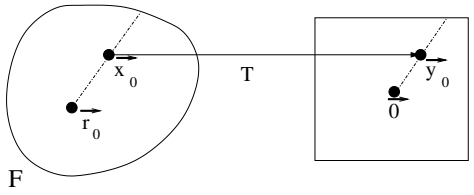
A more interesting type of approaches within this group are the so-called "Decoders". The emphasis of these approaches is to map chromosomes from the infeasible region into the feasible region of the problem to solve. In some cases, special operators have also been designed in order to produce offspring that lie on the boundary between the feasible and the infeasible region.

Special representations and operators

A more intriguing idea is to transform the whole feasible region into a different shape that is easier to explore. The most important approach designed along these lines are the "homomorphous maps" [Koziel & Michalewicz, 1999]. This approach performs a homomorphous mapping between an n-dimensional cube and a feasible search space (either convex or non-convex). The main idea of this approach is to transform the original problem into another (topologically equivalent) function that is easier to optimize by an evolutionary algorithm.

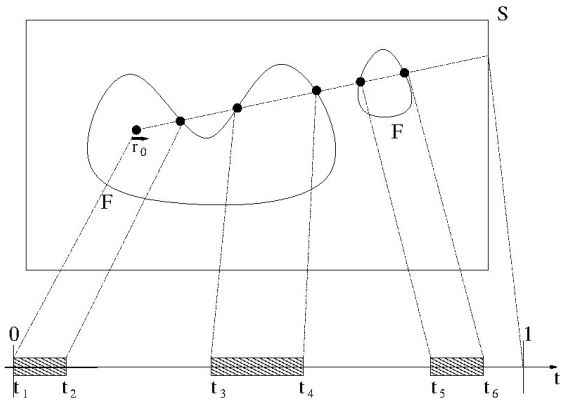
Special representations and operators

The convex case is the following:



Special representations and operators

The non-convex case is the following:



Special representations and operators

The Homomorphous Maps (HM) was for some time, the most competitive constraint-handling approach available (until the publication of Stochastic Ranking). However, the implementation of the algorithm is more complex, and the experiments reported required a high number of fitness function evaluations $(1,400,000)$.

Special representations and operators

The version of HM for convex feasible regions is very efficient. However, the version for non-convex feasible regions requires a parameter v and a binary search procedure to find the intersection of a line with the boundary of the feasible region.

Repair algorithms

This sort of approach consists in devising a procedure (or mechanism) that allows to transform an infeasible solution into a feasible one (i.e., we "repair" the infeasible individual). Such a repaired version can be used either for evaluation only, or it can also replace (with some probability) the original individual in the population.

Repair algorithms

Liepins and co-workers [1990,1991] showed, through an empirical test of EA performance on a diverse set of constrained-combinatorial optimization problems, that a repair algorithm is able to surpass other approaches in both speed and performance.

Repair algorithms

GENOCOP III [Michalewicz & Nazhiyath, 1995] also uses repair algorithmss. The idea is to incorporate the original GENOCOP system [Michalewicz & Janikow, 1991] (which handles only linear constraints) and extend it by maintaining two separate populations, where results in one population influence evaluations of individuals in the other population. The first population consists of the so-called search points which satisfy linear constraints of the problem; the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators. The second population consists of feasible reference points. Since these reference points are already feasible, they are evaluated directly by the objective function, whereas search points are "repaired" for evaluation.

Repair algorithms

Xiao and co-workers [1995] used a repair algorithm to transform an infeasible path of a robot trying to move between two points in the presence of obstacles, so that the path would become feasible (i.e., collision-free). The repair algorithm was implemented through a set of carefully designed genetic operators that used knowledge about the domain to bring infeasible solutions into the feasible region in an efficient way. Other authors that have used repair algorithms are Orvosh and Davis [1994], Mühlenbein [1992], Le Riche and Haftka [1994], and Tate and Smith [1995].

Repair algorithms

There are no standard heuristics for the design of repair algorithms: normally, it is possible to use a greedy algorithm (i.e., an optimization algorithm that proceeds through a series of alternatives by making the best decision, as computed locally, at each point in the series), a random algorithm or any other heuristic which would guide the repair process. However, the success of this approach relies mainly on the ability of the user to come up with such a heuristic.

Repair algorithms

Another interesting aspect of this technique is that normally an infeasible solution that is repaired is only used to compute its fitness, but the repaired version is returned to the population only in certain cases (using a certain probability). The question of replacing repaired individuals is related to the so-called Lamarckian evolution, which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the chromosome.

Repair algorithms

Some researchers like Liepins and co-workers [1990,1991] have taken the *never* replacing approach (that is, the repaired version is never returned to the population), while other authors such as Nakano [1991] have taken the *always* replacing approach. Orvosh and Davis [1993,1994] reported a so-called 5% rule for combinatorial optimization problems, which means that EAs (applied to combinatorial optimization problems) with a repairing procedure provide the best result when 5% of the repaired chromosomes replace their infeasible originals. Michalewicz [1996] has reported, however, that a 15% replacement rule seems to be the best choice for numerical optimization problems with nonlinear constraints.

Repair algorithms

When an infeasible solution can be easily (or at least at a low computational cost) transformed into a feasible solution, repair algorithms are a good choice. However this is not always possible and in some cases repair operators may introduce a strong bias in the search, harming the evolutionary process itself [Smith & Tate, 1993].

Repair algorithms

Furthermore, this approach is problem-dependent, since a specific repair algorithm has to be designed for each particular problem. Also, in its early days, this sort of approach was mostly used in combinatorial optimization problems. However, in recent years, this has become a relatively active research area (see for example [Salcedo-Sanz, 2009]).

Separation of constraints and objectives

Unlike penalty functions which combine the value of the objective function and the constraints of a problem to assign fitness, these approaches handle constraints and objectives separately. Some examples:

- **Coevolution**: Use two populations that interact with each other and have encounters [Paredis, 1994].

Separation of constraints and objectives

- **Superiority of feasible points**: The idea is to assign always a higher fitness to feasible solutions [Powell & Skolnick, 1993; Deb, 2000].
- **Behavioral memory**: Schoenauer and Xanthakis [1993] proposed to satisfy, sequentially, the constraints of a problem.

Separation of constraints and objectives

- **Use of multiobjective optimization concepts:** The main idea is to redefine the single-objective optimization of $f(\vec{x})$ as a multiobjective optimization problem in which we will have $m+1$ objectives, where m is the total number of constraints. Then, any multi-objective evolutionary algorithm can be adopted [Coello et al., 2002; Deb, 2001]. Note however, that the use of multiobjective optimization is not straightforward, and several issues have to be taken into consideration.

Separation of constraints and objectives

This last type of approach has been very popular in the last few years. For example:

- Surry & Radcliffe [1997] proposed COMOGA (Constrained Optimization by Multiobjective Optimization Genetic Algorithms) where individuals are Pareto-ranked based on the sum of constraint violation. Then, solutions can be selected using binary tournament selection based either on their rank or their objective function value.

Separation of constraints and objectives

- Zhou et al. [2003] proposed a ranking procedure based on the Pareto Strength concept (introduced in SPEA) for the bi-objective problem, i.e. to count the number of individuals which are dominated for a given solution. Ties are solved by the sum of constraint violation (second objective in the problem). The Simplex crossover (SPX) operator is used to generate a set of offspring where the individual with the highest Pareto strength and also the solution with the lowest sum of constraint violation are both selected to take part in the population for the next generation.

Separation of constraints and objectives

- Venkatraman and Yen [2005] proposed a generic framework divided in two phases: The first one treats the NLP as a constraint satisfaction problem i.e. the goal is to find at least one feasible solution. To achieve that, the population is ranked based only on the sum of constraint violation. The second phase starts when the first feasible solution was found. At this point, both objectives (original objective function and the sum of constraint violation) are taken into account and nondominated sorting [Deb, 2002] is used to rank the population.

Separation of constraints and objectives

- Hernandez et al. [2004] proposed an approach named IS-PAES which is based on the Pareto Archive Evolution Strategy (PAES) originally proposed by Knowles and Corne [2000]. IS-PAES uses an external memory to store the best set of solutions found. Furthermore, it adopts a shrinking mechanism to reduce the search space. The multiobjective concept is used in this case as a secondary criterion (Pareto dominance is used only to decide whether or not a new solution is inserted in the external memory).

Separation of constraints and objectives

Possible problems of the use of MO concepts:

Runarsson and Yao [2005] presented a comparison of two versions of Pareto ranking in constraint space: (1) considering the objective function value in the ranking process and (2) without considering it. These versions were compared against a typical over-penalized penalty function approach. The authors found that the use of Pareto Ranking leads to bias-free search, then, they concluded that it causes the search to spend most of the time searching in the infeasible region; therefore, the approach is unable to find feasible solutions (or finds feasible solutions with a poor value of the objective function).

Separation of constraints and objectives

Possible problems of the use of MO concepts:

Note however, that "pure" Pareto ranking is rarely used as a mechanism to handle constraints, since some bias is normally introduced to the selection mechanism (when a constraint is satisfied, it makes no sense to keep using it in the Pareto dominance relationship).

Hybrid methods

Within this category, we consider methods that are coupled with another technique (either another heuristic or a mathematical programming approach). Examples:

- Adeli and Cheng [1994] proposed a hybrid EA that integrates the penalty function method with the primal-dual method. This approach is based on sequential minimization of the Lagrangian method.

Hybrid methods

- Kim and Myung [1997] proposed the use of an evolutionary optimization method combined with an augmented Lagrangian function that guarantees the generation of feasible solutions during the search process.
- **Constrained optimization by random evolution (CORE)**: This is an approach proposed by Belur [1997] which combines a random evolution search with Nelder and Mead's method [1965].

Hybrid methods

- **Ant System (AS)**: The main AS algorithm is a multi-agent system where low level interactions between single agents (i.e., artificial ants) result in a complex behavior of the whole ant colony. Although mainly used for combinatorial optimization, AS has also been successfully applied to numerical optimization [Bilchev & Parmee, 1995; Leguizamon, 2004]. Some of the recent research in this area focuses on the exploration of the boundary between the feasible and infeasible regions [Leguizamon & Coello, 2006].

Hybrid methods

- **Simulated Annealing (SA)**: Wah & Chen [2001] proposed a hybrid of SA and a genetic algorithm (GA). The first part of the search is guided by SA. After that, the best solution is refined using a GA. To deal with constraints, Wah & Chen use Lagrangian Multipliers.

Hybrid methods

- **Artificial Immune System (AIS)**: Hajela and Lee [1996] proposed a GA hybridized with an AIS (based on the negative selection approach). The idea is to adopt as antigens some feasible solutions and evolve (in an inner GA) the antibodies (i.e., the infeasible solutions) so that they are "similar" (at a genotypic level) to the antigens.

Hybrid methods

- **Cultural Algorithms**: In this sort of approach, the main idea is to preserve beliefs that are socially accepted and discard (or prune) unacceptable beliefs. The acceptable beliefs can be seen as constraints that direct the population at the micro-evolutionary level. Therefore, constraints can influence directly the search process, leading to an efficient optimization process.

Hybrid methods

- In other words, when using cultural algorithms, some sort of knowledge is extracted during the search process and is used to influence the evolutionary operators as to allow a more efficient search. The first versions of cultural algorithms for constrained optimization had some memory handling problems [Chung & Reynolds, 1996], but later on, they were improved using spatial data structures that allowed to handle problems with any number of decision variables [Coello & Landa, 2002].

Recent Ideas: An Ensemble of Constraint-Handling Techniques

Mallipeddi and Suganthan [2010] proposed the use of an ensemble of constraint-handling techniques. The idea is to have several constraint-handling technique, each with its own population and parameters. Each population produces its offspring and evaluates them. However, the offspring compete not only against its own population, but also against the others. Thus, a certain offspring could be rejected by its population, while being accepted by another population. This intends the automate the selection of the best constraint-handling technique for a certain problem, based on the fact that none of them will be best in all cases (remember the No-Free-Lunch theorem!).

Recent Ideas: Use of Gradient-Based Information

There have been a few proposals that incorporate gradient information, either from the fitness values or from the constraints. For example, Sun and Garibaldi [2010] proposed an Estimation of Distribution Algorithm combined with a gradient-based local optimizer. Handoko et al. [2010] proposed a memetic algorithm that combines a genetic algorithm, sequential quadratic programming with second-order functional approximations, and a technique for modeling the feasibility region through the use of support vector machines. Hamza et al. [2014] proposed a consensus-based variant of a genetic algorithm which is combined with sequential quadratic programming, but using gradient information from the constraints.

Recent Ideas: Memetic Viability

Maesani et al. [2016] proposed the use of a notion called *viability evolution*, which emphasizes the elimination of solutions that do not satisfy viability criteria (i.e., boundaries on objectives and constraints). Such boundaries are adapted during the search in a way analogous to other approaches such as ASCHEA. However, in this case, this approach (which is based on the use of the covariance matrix adaptation evolution strategy (CMA-ES)) produces a population of local search engines which can be recombined using differential evolution. An interesting aspect of this approach is that it uses an adaptive scheduler that toggles between exploration and exploitation by selecting to advance one of the local search units (or individuals) or to recombine them.

Test Functions

Michalewicz and Schoenauer [1996] proposed a set of test functions, which was later expanded by Runarsson and Yao [2000]. The current set contains 13 test functions. These test functions contain characteristics that are representative of what can be considered "difficult" global optimization problems for an evolutionary algorithm.

Test Functions

Note however, that many other test functions exist. See for example:

Mezura Montes, Efrén and Coello Coello, Carlos A., **What Makes a Constrained Problem Difficult to Solve by an Evolutionary Algorithm**, Technical Report EVOCINV-01-2004, Evolutionary Computation Group at CINVESTAV, Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México, February 2004.

C. A. Floudas and P. M. Pardalos, **A Collection of Test Problems for Constrained Global Optimization Algorithms**, Number 455 in Lecture Notes in Computer Science. Springer-Verlag, 1990.

Christodoulos A. Floudas et al. (editors), **Handbook of Test Problems in Local and Global Optimization**, Kluwer Academic Publishers, Dordrecht, 1999.

Test Functions (Current Benchmark)

Problem	n	Type of function	ρ	LI	NI	LE	NE
g01	13	quadratic	0.0003%	9	0	0	0
g02	20	nonlinear	99.9973%	1	1	0	0
g03	10	nonlinear	0.0026%	0	0	0	1
g04	5	quadratic	27.0079%	0	6	0	0
g05	4	nonlinear	0.0000%	2	0	0	3
g06	2	nonlinear	0.0057%	0	2	0	0
g07	10	quadratic	0.0000%	3	5	0	0
g08	2	nonlinear	0.8581%	0	2	0	0
g09	7	nonlinear	0.5199%	0	4	0	0
g10	8	linear	0.0020%	3	3	0	0
g11	2	quadratic	0.0973%	0	0	0	1
g12	3	quadratic	4.7697%	0	9^3	0	0
g13	5	nonlinear	0.0000%	0	0	1	2

Test Functions

Additional test functions have been proposed, and a new benchmark, consisting of 24 test functions (which include the 13 indicated in the previous slide) is now more popular. See:

J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. Coello Coello, K. Deb, **Problem Definitions and Evaluation Criteria for the CEC 2006, Special Session on Constrained Real-Parameter Optimization**, Technical Report, Nanyang Technological University, Singapore, 2006.

There is also a set of test problems that were introduced at the *2010 World Congress on Computational Intelligence* (WCCI'2010). This set consists of 18 scalable test problems.

Test Case Generators

Michalewicz [2000] proposed a Test Case Generator for constrained parameter optimization techniques. This generator allows to build test problems by varying several features such as: dimensionality, multimodality, number of constraints, connectedness of the feasible region, size of the feasible region with respect to the whole search space and ruggedness of the objective function.

Test Case Generators

The first version of this test problems generator had some problems because the functions produced were symmetric. This motivated the development of a new version called TCG-2 [Schmidt et al., 2000].

Some Recommendations

- Study (and try first) traditional mathematical programming techniques (e.g., gradient-based methods, Nelder-Mead, Hooke-Jeeves, etc.).
- If interested in numerical optimization, try evolution strategies or differential evolution, instead of using genetic algorithms. Also, the combination of parents and offspring in the selection process tends to produce better performance.
- Pay attention to diversity. Keeping populations in which every individual is feasible is not always a good idea.
- Normalizing the constraints of the problem is normally a good idea.

Current Research Topics

- New constraint-handling approaches (e.g., based on multiobjective optimization concepts).
- Large-scale constrained optimization [Aguilar-Justo & Mezura-Montes, 2016].
- Use of voting schemes as a way of incorporating preferences for allowing the exploration of a portion of the infeasible region [Yu, 2014].
- Old constraint-handling techniques with new search engines (e.g., differential evolution, particle swarm optimization, ant colony, etc.).
- Constraint-handling techniques for multi-objective evolutionary algorithms [Yen, 2009].

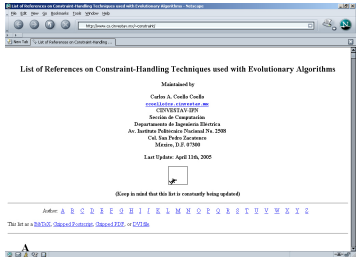
Current Research Topics

- Self-adaptation mechanisms for constrained optimization [Brest, 2009].
- Equality constraint-handling in multi-objective optimization.
- Time complexity analysis of evolutionary algorithms used for solving constrained problems (particularly, the role of penalty factors in the time complexity of an EA) [Zhou & He, 2007].
- Hybrids of EAs with mathematical programming techniques (e.g., evolution strategy + simplex, use of Lagrange multipliers, etc.).
- Approaches that reduce the number of objective function evaluations performed (e.g., surrogate models, fitness inheritance, fitness approximation).

Current Research Topics

- Constrained robust optimization [Tagawa and Miyanaga, 2017].
- Test function generators (how to build them and make them reliable? Test functions available online?).
- New metrics that allow us to evaluate the online performance of a constraint-handling technique and to characterize constrained search spaces (see for example the notion of *violation landscape* proposed in [Malan, 2015]).
- Stopping criteria for constrained optimization using evolutionary algorithms [Zielinski, 2009].
- Special operators for exploring the boundary between the feasible and infeasible regions [Leguizamón & Coello, 2009].
- Dynamic constraints [Nguyen & Yao, 2012].

To know more about constraint-handling techniques used with EAs


List of References on Constraint-Handling Techniques used with Evolutionary Algorithms

Maintained by

Carlos A. Coello Coello

Last Update: April 11th, 2005

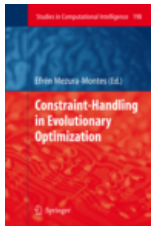
(Keep in mind that this list is constantly being updated)

Please visit our constraint-handling repository located at:

http://www.cs.cinvestav.mx/~constraint

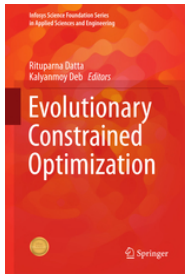
The repository currently (as of April 13th, 2018) had **1438** bibliographic references.

Suggested Readings



Efrén Mezura-Montes (editor), **Constraint-Handling in Evolutionary Optimization**, Springer, 2009, ISBN: 978-3-642-00618-0.

Suggested Readings



Rituparna Datta and Kalyanmoy Deb (editors), **Evolutionary Constrained Optimization**, Springer, 2015, ISBN: 978-81-322-2183-8.

Suggested Readings

- Efrén Mezura-Montes and Carlos A. Coello Coello, **Constraint-Handling in Nature-Inspired Numerical Optimization: Past, Present and Future**, *Swarm and Evolutionary Computation*, Vol. 1, No. 4, pp. 173–194, December 2011.

Suggested Readings

- Zbigniew Michalewicz and Marc Schoenauer, **Evolutionary Algorithms for Constrained Parameter Optimization Problems**, *Evolutionary Computation*, Vol. 4, No. 1, pp. 1–32, 1996.
- Carlos A. Coello Coello, **Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art**, *Computer Methods in Applied Mechanics and Engineering*, Vol. 191, No. 11–12, pp. 1245–1287, January 2002.

Suggested Readings

- Yuren Zhou and Jun He, **A Runtime Analysis of Evolutionary Algorithms for Constrained Optimization Problems**, *IEEE Transactions on Evolutionary Computation*, Vol. 11, No. 5, pp. 608–619, October 2007.
- Thomas P. Runarsson and Xin Yao, **Stochastic Ranking for Constrained Evolutionary Optimization**, *IEEE Transactions on Evolutionary Computation*, **4**(3):284–294, September 2000.

Suggested Readings

- Martin Schmidt and Zbigniew Michalewicz, **Test-Case Generator TCG-2 for Nonlinear Parameter Optimization**, in M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Proceedings of 6th Parallel Problem Solving From Nature (PPSN VI)*, pp. 539–548, Heidelberg, Germany, September 2000, Springer-Verlag. Lecture Notes in Computer Science Vol. 1917.
- Alice E. Smith and David W. Coit, **Constraint Handling Techniques–Penalty Functions**, in Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C 5.2. Oxford University Press and Institute of Physics Publishing, 1997.

Suggested Readings

- R. Mallipeddi and P. N. Suganthan, **Problem Definitions and Evaluation Criteria for the CEC 2010 Competition on Constrained Real-Parameter Optimization**, Technical Report, Nanyang Technological University, Singapore, 2010. Available at: `http://www.ntu.edu.sg/home/EPNSugan/`.
- Rammohan Mallipeddi and Ponnuthurai N. Suganthan, **Ensemble of Constraint Handling Techniques**, *IEEE Transactions on Evolutionary Computation*, Vol. 14, No. 4, pp. 561-579, August 2010.

Suggested Readings

- Trung Thanh Nguyen and Xin Yao, **Continuous Dynamic Constrained Optimization—The Challenges**, *IEEE Transactions on Evolutionary Computation*, Vol. 16, No. 6, pp. 769–786, December 2012.
- Oliver Kramer, **A Review of Constraint-Handling Techniques for Evolution Strategies**, *Applied Computational Intelligence and Soft Computing*, Vol. 2010, Article ID 185063, No. 1, pp. 1–11, January 2010.
- Sancho Salcedo-Sanz, **A Survey of Repair Methods Used as Constraint Handling Techniques in Evolutionary Algorithms**, *Computer Science Review*, Vol. 3, No. 3, pp. 175–192, 2009.

Suggested Readings

- Erdong Yu, Qing Fei, Hongbin Ma and Qingbo Geng, **Improving Constraint Handling for Multiobjective Particle Swarm Optimization**, in *Proceedings of the 33rd Chinese Control Conference*, pp. 8622–8627, IEEE Press, Nanjing, China, July 28-30, 2014.
- Katherine M. Malan, Johannes F. Oberholzer and Andries P. Engelbrecht, **Characterising Constrained Continuous Optimisation Problems**, in *2015 IEEE Congress on Evolutionary Computation (CEC'2015)*, pp. 1351–1358, IEEE Press, Sendai, Japan, 25-28 May 2015, ISBN 978-1-4799-7492-4.
- Andrea Maesani, Giovanni Iacca and Dario Floreano, **Memetic Viability Evolution for Constrained Optimization**, *IEEE Transactions on Evolutionary Computation*, Vol. 20, No. 1, pp. 125–144, February 2016.

Suggested Readings

- Jianyong Sun and Jonathan M. Garibaldi, **A Novel Memetic Algorithm for Constrained Optimization**, in *2010 IEEE Congress on Evolutionary Computation (CEC'2010)*, pp. 549-556, IEEE Press, Barcelona, Spain, July 18–23, 2010.
- Stephanus Daniel Handoko, Chee Keong Kwoh and Yew-Soon Ong, **Feasibility Structure Modeling: An Effective Chaperone for Constrained Memetic Algorithms**, *IEEE Transactions on Evolutionary Computation*, Vol. 14, No. 5, pp. 740–758, October 2010.
- Noha M. Hamza, Ruhul A. Sarker, Daryl L. Essam, Kalyanmoy Deb and Saber M. Elsayed, **A Constraint Consensus Memetic Algorithm for Solving Constrained Optimization Problems**, *Engineering Optimization*, Vol. 46, No. 11, pp. 1447–1464, November 2014.