Constraint-Handling Techniques used with Evolutionary Algorithms

Carlos A. Coello Coello

CINVESTAV-IPN

Depto. de Ingeniería Eléctrica

Sección de Computación

Av. Instituto Politécnico Nacional No. 2508

Col. San Pedro Zacatenco

México, D. F. 07360, MEXICO

ccoello@cs.cinvestav.mx

Motivation

Traditional mathematical programming techniques used to solve constrained optimization problems have several limitations when dealing with the general non-linear 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

S

F

F

F
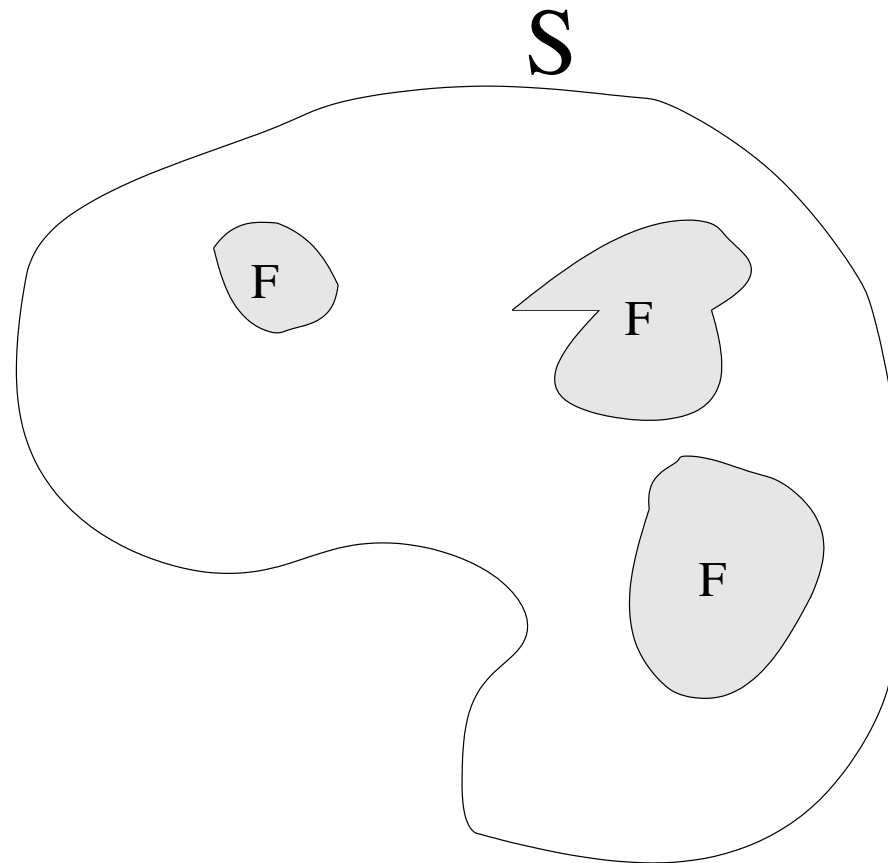
A Taxonomy of Constraint-Handling Approaches

- Penalty Functions

- Special representations and operators

- 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 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. 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:

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

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

- Recent 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. For example:

- 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):

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

  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). For example:

- 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).

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. More recent 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. 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.

Recent 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

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).

- 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.

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:

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

where

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

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:

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

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

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

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

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.

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. 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. 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]. The idea is to perform 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 Homomorphous Maps (HM) was the most competitive constraint-handling approach (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 $(1,400,000)$. 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.

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.

- **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.

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 the Nelder and Mead's simplex method [1965].

- **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].

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.

- **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. In other words, 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 [Chung & Reynolds, 1996; Coello & Landa, 2002].

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.

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

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 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.

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, S^2 or simplex, 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.

Current Research Topics

- New constraint-handling approaches.

- Old techniques with new search engines (e.g., differential evolution, particle swarm optimization, etc.).

- Hybrids of EAs with mathematical programming techniques (e.g., evolution strategy + simplex).

- Test function generators (how to build them and make them reliable? Test functions available online?).

- New metrics that allow the evaluate the online performance of a constraint-handling technique.

To know more about constraint-handling techniques used with EAs
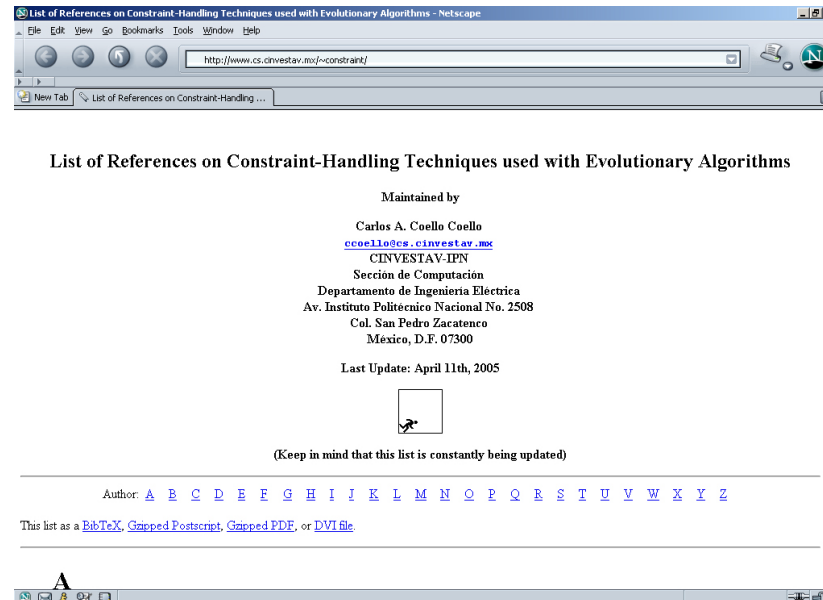
Please visit our constraint-handling repository located at:

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

The repository currently (as of April 15th, 2005) has **462** bibliographic references.

Suggested Readings

1. Zbigniew Michalewicz and Marc Schoenauer, "Evolutionary Algorithms for Constrained Parameter Optimization Problems", *Evolutionary Computation*, Vol. 4, No. 1, pp. 1–32, 1996.

2. 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.

3. Thomas P. Runarsson and Xin Yao, "Stochastic Ranking for Constrained Evolutionary Optimization", *IEEE Transactions on Evolutionary Computation*, **4**(3):284–294, September 2000.

4. 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.