Constraint-Handling in Evolutionary Algorithms

Jiří Kubalík

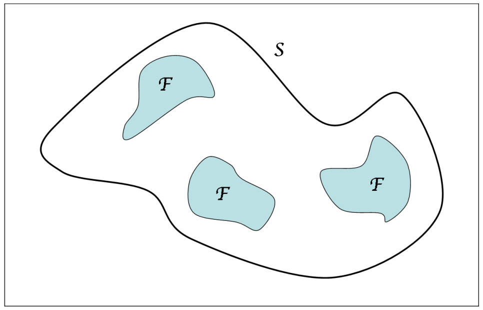
Czech Institute of Informatics, Robotics and Cybernetics
CTU Prague

http://cw.felk.cvut.cz/doku.php/courses/a0m33eoa/start

Motivation

The general nonlinear programming problem (NLP) can be formulated as solving the objective function

$$\begin{array}{rll} \text{Minimize} & f(x), & x = [x_1, \ldots, x_n]^T; \\ \text{subject to} & g_i(x) \leq 0, & i = 1, 2, ..., m; \\ & h_j(x) = 0, & j = 1, 2, ..., p; \end{array}$$



where

- x is a vector of n decision variables,
- each x_i, $i = 1, \ldots, n$ is bounded by lower and upper limits $x_i^{(L)} \leq x_i \leq x_i^{(U)}$, which define the search space $\mathcal{S}$,
- $\mathcal{F} \subseteq \mathcal{S}$ is the feasible region defined by m inequality and p equality constraints.

When solving NLP with EAs, equality constraints are usually transformed into inequality constraints of the form:

$$|h_j(x)| - \varepsilon \leq 0$$

where ε is the tolerance allowed.

Standard Evolutionary Algorithm

```
Begin
   t=0
   Initialize P(t)
   Evaluate P(t)
   while (not termination-condition) do
   begin
     t=t+1
     Select P(t) from P(t-1)
     Recombine
     Evaluate P(t)
   end
End
```

Evolutionary Algorithms (EAs) have been found successful in solving a wide variety of optimization problems.

However, **EAs are unconstrained search techniques**. Therefore, it is necessary to incorporate constraints into components of the EA (i.e. the fitness function and genetic operators).

Taxonomy of Presented Constraint-Handling Approaches

- Penalty functions
- Stochastic Ranking
- Special representations and operators
- Multiobjective optimization techniques

General Form of Fitness Function with Penalty Function

The idea of penalty functions is to **transform a constrained optimization problem into unconstrained one by adding certain value to the objective function** based on the amount of constraint violation present in the solution:

$$\psi(x) = f(x) + \sum_{i=1}^{m} r_i \times G_i + \sum_{j=1}^{p} r_j \times H_j$$

where $\psi(x)$ is the new objective function referred to as the **fitness function**, G_i and H_j are functions of the constraints violation $\left(g_i(x) \text{ and } h_j(x)\right)$, and r_i and r_j are positive constants called **penalty coefficients** or penalty factors.

A common form of G_i:

$$G_i = \text{max}(0, g_i(x))$$

A common form of H_j:

$$H_j = |h_j(x)|$$

or

$$H_j = \text{max}(0, g_j(x)), \text{ for } g_j \equiv |h_j(x)| - \varepsilon \leq 0$$

Types of Penalty Functions used with EAs

Four categories of penalty functions based on the way its parameters are being determined:

- Death penalty
- Static penalty
- Dynamic penalty
- **Adaptive penalty**

Death Penalty

The **rejection of infeasible individuals** 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.

- The approach is to iterate, generating a new point at each iteration, until a feasible solution is found.

Thus, no further calculations are necessary to estimate the degree of infeasibility of such a solution.

Death Penalty

The **rejection of infeasible individuals** 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.

- The approach is to iterate, generating a new point at each iteration, until a feasible solution is found.

Thus, no further calculations are necessary to estimate the degree of infeasibility of such a solution.

Criticism:

- No exploitation of the information from infeasible solutions.

  Search may "stagnate" in the presence of very small feasible regions.

  Not advisable, except in the case of problems in which the proportion of feasible region in the whole search space is fairly large.

- A variation that assigns a zero (or *very bad*) fitness to infeasible solutions may work better in practice.

Death Penalty: Relaxed

Fitness of an individual is determined using:

$$fitness_i(x) = \begin{cases} f(x) & \text{if the solution is feasible} \\ K(1 - \frac{s}{m+p}) & \text{otherwise} \end{cases}$$

where

- s is the number of constraints satisfied, and
- K is a large constant.

If an individual is infeasible,

- its fitness is always worse than a fitness of any other feasible individual and
- its fitness is the same as the fitness of all the individuals that violate the same number of constraints.

Static Penalty

Approaches in which the **penalty coefficients do not depend** on the current generation number, they remain constant during the entire evolution.

The approach proposed in [Homaifar94] defines **levels of violation** of the constraints (and **penalty coefficients** associated to them):

$$fitness(x) = f(x) + \sum_{i=1}^{m+p} (R_{k,i} \times (max[0, g_i(x)])^2)$$

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

Static Penalty

Approaches in which the **penalty coefficients do not depend** on the current generation number, they remain constant during the entire evolution.

The approach proposed in [Homaifar94] defines **levels of violation** of the constraints (and **penalty coefficients** associated to them):

$$fitness(x) = f(x) + \sum_{i=1}^{m+p} (R_{k,i} \times (max[0, g_i(x)])^2)$$

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

Criticism:

- The weakness of the method is the high number of parameters: for m constraints and l levels of violation for each, the method requires $m(2l + 1)$ parameters in total.

  For $m = 5$ and $l = 4$, we need 45 parameters!!!

  Clearly, the results are heavily parameter dependent.

- Presented method requires prior knowledge of the degree of constraint violation present in the problem (to define the levels of violation), which might not be easy to obtain in real-world applications.

- Penalty coefficients are difficult to generalize. They are, in general, problem-dependent.
- Anyway, it is not a good idea to keep the same penalty coefficient along the entire evolution. The population evolves, so why should the coefficients that bias the search direction be static?

Dynamic Penalty

Penalty functions in which the **current generation number is involved in the computation of the corresponding penalty coefficients**.

Typically, the penalty coefficients are defined in such a way that **they increase over time** pushing the search towards the feasible region.

The approach from [Joines94] evaluates individuals as follows:

$$fitness(x) = f(x) + (C \times t)^{\alpha} \times SVC(x)$$

where t is the generation number and C and α are user-defined constants; recommended values are $C = 0.5$ or 0.05, $\alpha = 1$ or **2**.

$SVC(x)$ is defined as:

$$SVC(x) = \sum_{i=1}^{m} G_i(x) + \sum_{j=1}^{p} H_j(x)$$

where $G_i(x)$ and $H_j(x)$ are functions of the constraints violation ($g_i(x)$ and $h_j(x)$).

Step-wise non-stationary penalty function increases the penalty proportionally to the generation number. The goal is to allow the GA to explore more of the search space before confining it to the feasible region.

Dynamic Penalty: Criticism

- It is difficult to derive good dynamic penalty functions in practice.

  The presented approach is sensitive to changes in values of α and C and there are no guidelines for choosing proper values for particular problem.

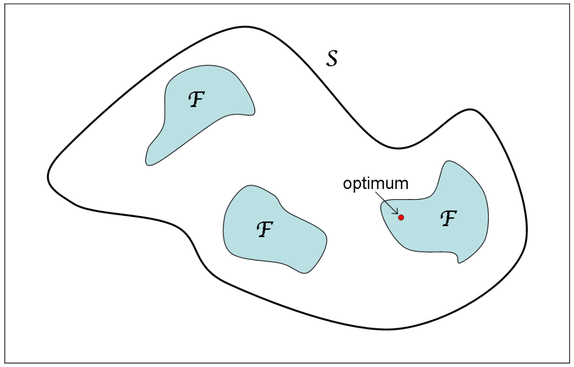
- If a bad penalty coefficient is chosen, the EA may converge to either non-optimal feasible solutions (if the penalty is too high) or to infeasible solutions (if the penalty is too low).

- No constraint normalization technique is used, thus certain constraint may undesirably dominate the whole penalty value.

Adaptive Penalty: Motivation

Let's assume the penalty fitness function of the following form:

$$\psi(x) = f(x) + r_g \times \sum_{i=1}^{m+p} G_i(x)^2$$



Deciding on an optimal (or near-optimal) value r_g is a difficult optimization problem itself.

- If r_g is **too small**, an infeasible solution may not be penalized enough. Hence, infeasible solutions may be evolved by an EA.
- If r_g is **too large**, a feasible solution is likely to be found, but could be of a poor quality.

  A large r_g discourages the exploration of infeasible regions.

  This is inefficient for problems where feasible regions in the whole search space are disjoint and/or the constraint optimum lies close to the boundary of the feasible domain.

Reasonable exploration of infeasible regions may act as bridges connecting feasible regions.

How much exploration of infeasible regions ($r_g =?$) is reasonable?

- It is problem dependent.
- Even for the same problem, different stages of evol. search may require different r_g values.

Adaptive Penalty

- GA with non-linear penalty function.
- Adaptive Segregational Constraint Handling EA (ASCHEA).
- Stochastic Ranking.

GA with Non-linear Penalty Function

The method introduced in [Hadj-Alouane92] uses adaptive penalty function that takes a feedback from the search process

$$fitness_i(x) = f_i(x) + r_g(t) \sum_{j=1}^{m+p} G_i(x)^2$$

where $r_g(t)$ is updated every generation according to the following rule:

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

where

- case $\#1$ – the situation where the best individual in the last k generations was **always feasible**;
- case $\#2$ – the situation where the best individual in the last k generations was **never feasible**;
- $\beta_1, \beta_2 > 1$, and $\beta_1 \neq \beta_2$ (to avoid cycling).

Remarks:

- The penalty component for the generation $t+1$ is decreased if the feasible region **was** effectively sampled within last k generations;

- The penalty component for the generation $t + 1$ is increased if the feasible region **was not** effectively sampled within last k generations;
- It tries to avoid having either an all-feasible or an all-infeasible population.
- The problem is how to choose a proper generational gap (k) and the values of β_2 and β_1.

Adaptive Segregational Constraint Handling EA - ASCHEA

The main idea in ASCHEA [Hamida00] is to maintain both feasible and infeasible individuals in the population, at least when it seems necessary.

It proposes adaptive mechanisms at the population level for constraint optimization based on three main components:

1. **Adaptive penalty function** – takes care of the penalty coefficients according to the **proportion of feasible individuals** in the current population.
2. **Constraint-driven mate selection** – used to **mate feasible individuals with infeasible ones** and thus explore the region around the boundary of the feasible domain.
3. **Segregational replacement strategy** – used to favor a given number of feasible individuals in the population.

ASCHEA: Adaptive Penalty

Let's assume the penalty function of the following form:

$$penal(x) = \alpha \sum_{i=1}^{m+p} G_i(x)$$

The penalty coefficient α is adapted based on the desired proportion of feasible solutions in the population, τ_{target}, and the actual proportion at generation t, τ_t:

$$\begin{array}{rl} \text{if}(\tau_t > \tau_{target}) & \alpha(t+1) = \alpha(t)/fact \\ \text{otherwise} & \alpha(t+1) = \alpha(t) * fact \end{array}$$

where $fact > 1$ is a user-defined parameter, a recommended value is around 1.1.

A recommended value of τ_{target} is around 0.6.

ASCHEA: Constraint-driven Mate Selection

Selection mechanism chooses the **mate of feasible individuals to be infeasible**.

- Only applied when too few (w.r.t τ_{target}) feasible individuals are present in the population.

More precisely, to select the mate x_2 for a first parent x_1:

if($0 < \tau_t < \tau_{target}$)and($x_1$ is feasible)	select x_2 among infeasible solutions only
otherwise	select x_2 according to fitness

ASCHEA: Segregational Replacement

Deterministic replacement mechanism used in ES-like scheme that should further **enhance the chances of survival of feasible individuals**.

Assume a population of μ parents, from which λ offspring are generated. Depending on the replacement scheme

- μ individuals out of λ offspring in case of the (μ, λ)-ES, or
- μ individuals out of λ offspring plus μ parents in case of the $(\mu + \lambda)$-ES

are selected to the new population in the following way:

1. First, **feasible solutions are selected without replacement** based on their fitness, until $\tau_{select} * \mu$ have been selected, or no more feasible solution is available.
2. The population is then filled in using **standard deterministic selection on the remaining individuals**, based on the penalized fitness.

Thus, a user-defined proportion of τ_{select} feasible solutions is considered superior to all infeasible solutions.

A recommended value of τ_{select} is around 0.3.

ASCHEA: Niching

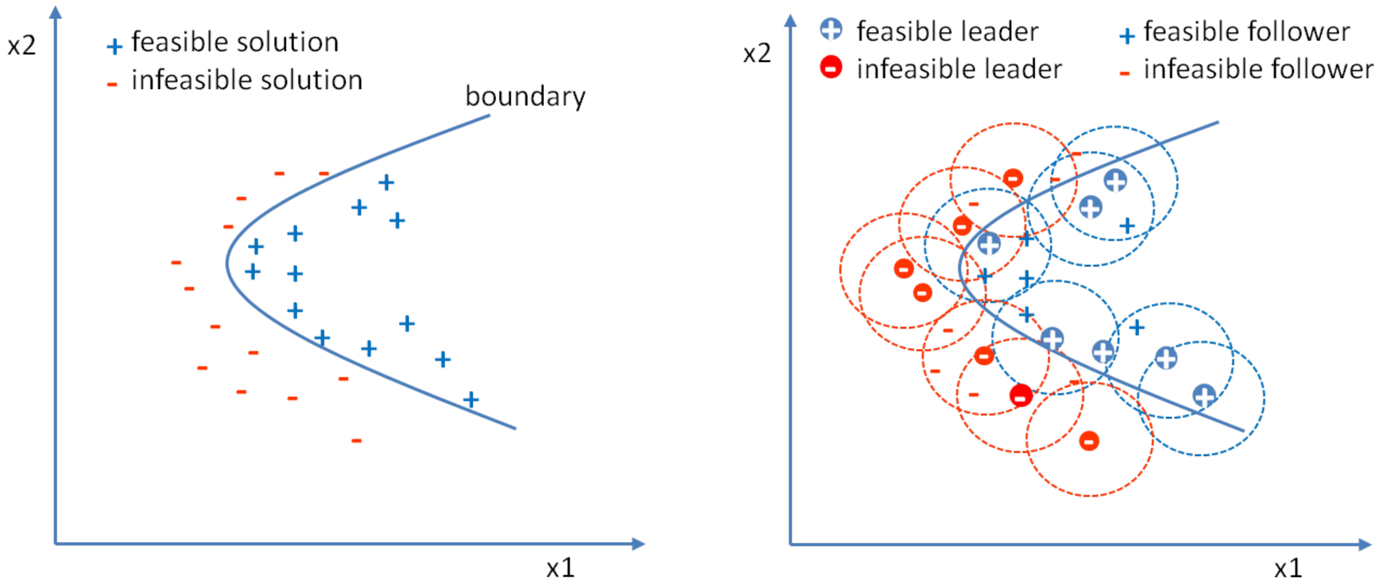
Niching – helps to better handle multimodal functions.

- a niche defined as a hypersphere around a good individual of a diameter r
- a niche has its **capacity**, $niche_{capacity}$, defining the maximal number of **leaders** in a niche (including its central individual)
- other individuals falling within the niche, called **followers**, are discarded from further selections

input: population sorted from the best to the worst individual)

for $i = 1$ to μ
if $\overrightarrow{x_i} \notin followers$
    add $\overrightarrow{x_i}$ to $leaders$
    $nbLeaders = 1$
    for $j = i + 1$ to μ
    if $\overrightarrow{x_j} \notin followers$ and $distance(\overrightarrow{x_i}, \overrightarrow{x_j}) < r$
        if $nbLeaders < niche_{capacity}$
            $nbLeaders++$
        else
            add $\overrightarrow{x_j}$ to $followers$

ASCHEA: Niching with $niche_{capacity} = 2$



Niching is used in segregational replacement

- the replacement is first applied on the leaders,
- then on the followers, if necessary.

ASCHEA: Niching

Adaptation of the niche radius r

- τ_{clear} is a desired proportion of *leaders* (or *followers*) in the population, typically $\tau_{clear} = 0.4$
- $\tau_{leaders}$ and $\tau_{followers}$ are the proportions of leaders and followers in the population at the generation t.

$$\begin{array}{ll} \text{if}(\tau_{leaders} > \tau_{clear}) & r(t+1) = r(t) * fact \\ \text{else if}(\tau_{followers} > \tau_{clear}) & r(t+1) = r(t)/fact \end{array}$$

where $fact > 1$ is a user-defined parameter, a recommended value is around 1.1.

ASCHEA: Conclusions

- **Feasibility elitism** – as soon as a feasible individual appears, it can only disappear from the population by being replaced by a better feasible solution, even if the penalty coefficient reaches very small value.
- **Constraint-driven mate selection** accelerates the movement toward the feasible region of infeasible individuals, and helps to explore the region close to the boundary of the feasible domain.
- **Adaptability** – the penalty adaptation as well as the constraint-driven mate selection are activated based on the actual proportion of feasible solutions in the population.

Stochastic Ranking: What Penalty Methods Do?

Let's assume the penalty fitness function of the following form:

$$\psi(x) = f(x) + r_g \times \sum_{i=1}^{m+p} G_i(x)^2$$

For a given penalty coefficient $r_g > 0$, let the ranking of λ individuals be

$$\psi(x_1) \leq \psi(x_2) \leq \cdots \leq \psi(x_\lambda), \tag{1}$$

For any given adjacent pair i and $i+1$ in the ranked order

$$f_i + r_g G_i \leq f_{i+1} + r_g G_{i+1} \text{ where } f_i = f(x_i) \text{ and } G_i = G(x_i) \tag{2}$$

we define so called critical penalty coefficient

$$\check{r}_i = (f_{i+1} - f_i)/(G_i - G_{i+1}) \text{ for } G_i \neq G_{i+1} \tag{3}$$

Stochastic Ranking: What Penalty Methods Do?

For given choice of $r_g \geq 0$, there are three different cases which may give rise to inequality (2):

1. $f_i \leq f_{i+1}$ and $G_i \geq G_{i+1}$: Objective function plays a dominant role in determining the inequality and the value of r_g should be $0 < r_g < \check{r}_i$.
2. $f_i \geq f_{i+1}$ and $G_i < G_{i+1}$: Penalty function plays a dominant role in determining the inequality and the value of r_g should be $0 < \check{r}_i < r_g$.
3. $f_i < f_{i+1}$ and $G_i < G_{i+1}$: The comparison is nondominated and $\check{r}_i < 0$. Neither the objective nor the penalty function can determine the inequality by itself.

Stochastic Ranking: What Penalty Methods Do?

For given choice of $r_g \geq 0$, there are three different cases which may give rise to inequality (2):

1. $f_i \leq f_{i+1}$ and $G_i \geq G_{i+1}$: Objective function plays a dominant role in determining the inequality and the value of r_g should be $0 < r_g < \check{r}_i$.

   Ex.:

$$
\begin{array}{lll}
f_i = 10, & G_i = 7 & \\
f_{i+1} = 20 & G_{i+1} = 5 & \\
\multicolumn{3}{l}{\check{r}_i = (20-10)/(7-5) = 5 \quad \Longrightarrow \quad 0 < r_g < 5} \\
\hline
r_g = 4: & 38 \leq 40 & \text{the inequality (2) holds} \\
r_g = 6: & 52 \not\leq 50 & \text{the inequality (2) does not hold}
\end{array}
$$

2. $f_i \geq f_{i+1}$ and $G_i < G_{i+1}$: Penalty function plays a dominant role in determining the inequality and the value of r_g should be $0 < \check{r}_i < r_g$.

3. $f_i < f_{i+1}$ and $G_i < G_{i+1}$: The comparison is nondominated and $\check{r}_i < 0$. Neither the objective nor the penalty function can determine the inequality by itself.

Stochastic Ranking: What Penalty Methods Do?

For given choice of $r_g \geq 0$, there are three different cases which may give rise to inequality (2):

1. $f_i \leq f_{i+1}$ and $G_i \geq G_{i+1}$: Objective function plays a dominant role in determining the inequality and the value of r_g should be $0 < r_g < \check{r}_i$.

2. $f_i \geq f_{i+1}$ and $G_i < G_{i+1}$: Penalty function plays a dominant role in determining the inequality and the value of r_g should be $0 < \check{r}_i < r_g$.

   Ex.:

$$\begin{array}{lll} f_i = 20, & G_i = 5 & \\ f_{i+1} = 10 & G_{i+1} = 7 & \\ \check{r}_i = (10-20)/(5-7) = 5 & \Longrightarrow & 5 < r_g \\ \hline r_g = 4: & 40 \not\leq 38 & \text{the inequality (2) does not hold} \\ r_g = 6: & 50 \leq 52 & \text{the inequality (2) holds} \end{array}$$

3. $f_i < f_{i+1}$ and $G_i < G_{i+1}$: The comparison is nondominated and $\check{r}_i < 0$. Neither the objective nor the penalty function can determine the inequality by itself.

Stochastic Ranking: What Penalty Methods Do?

For given choice of $r_g \geq 0$, there are three different cases which may give rise to inequality (2):

1. $f_i \leq f_{i+1}$ and $G_i \geq G_{i+1}$: Objective function plays a dominant role in determining the inequality and the value of r_g should be $0 < r_g < \check{r}_i$.
2. $f_i \geq f_{i+1}$ and $G_i < G_{i+1}$: Penalty function plays a dominant role in determining the inequality and the value of r_g should be $0 < \check{r}_i < r_g$.
3. $f_i < f_{i+1}$ and $G_i < G_{i+1}$: The comparison is nondominated and $\check{r}_i < 0$. Neither the objective nor the penalty function can determine the inequality by itself.

   Ex.:

   $$\begin{array}{ll} f_i = 10, & G_i = 5 \\ f_{i+1} = 20 & G_{i+1} = 7 \\ \check{r}_i = (20-10)/(5-7) = -5 & \Longrightarrow \quad \text{the inequality (2) holds for all } r_g > 0 \end{array}$$

Stochastic Ranking: What Penalty Methods Do?

The value of r_g has no impact on the inequality (2) when nondominant and feasible individuals are compared.

The value of r_g is critical in the first two cases.

For the entire population, the chosen value of r_g used for comparisons will determine the fraction of individuals dominated by the objective and penalty functions.

It has to be within a certain range $\underline{r}_g < r_g < \overline{r}_g$

1. $\overline{r}_g$ is the **maximum critical penalty** coefficient computed from adjacent individuals ranked only according to the objective function.
2. $\underline{r}_g$ is the **minimum critical penalty** coefficient computed from adjacent individuals ranked only according to the penalty function.

Both **bounds are problem dependent** and may **vary from generation to generation** as they are also determined by the current population.

Stochastic Ranking: What Penalty Methods Do?

There are three categories of r_g values

1. $r_g < \underline{r}_g$: *Underpenalization* – All comparisons are based only on the fitness function.
2. $r_g > \overline{r}_g$: *Overpenalization* – All comparisons are based only on the penalty function.
3. $\underline{r}_g < r_g < \overline{r}_g$: All comparisons are based on a combination of objective and penalty functions.

   This is what a good constraint-handling technique should do – **to balance between preserving feasible individuals and rejecting infeasible ones**.

   But the optimal r_g is hard to determine.

All penalty methods can be classified into one of the above three categories. Some methods may fall into different categories during different stages in search

Stochastic Ranking: Realization

Stochastic Ranking [Runarsson00] characteristics:

- The idea of this approach is that **the balance between objective and penalty functions is achieved directly and explicitly**.
- It does not require explicit definition of the penalty coefficient r_g value.

  Instead, it requires a **user-defined parameter P_f, which determines the balance** between the objective function and the penalty function.

Stochastic ranking realization: **Bubble-sort-like procedure**

- The population is sorted using an algorithm similar to bubble-sort.
- Parameter P_f specifies a **probability of using only the objective function** for comparisons of infeasible solutions.

  If both individuals are feasible then the probability of comparing them according to the objective function is 1. Otherwise, it is P_f.

  The reminder of the comparisons are realized based on the sum of constraint violation.

  Recommended range of P_f values is $(0.4, 0.5)$

The P_f introduces the stochastic component to the ranking process, so that **some solutions may get a good rank even if they are infeasible**.

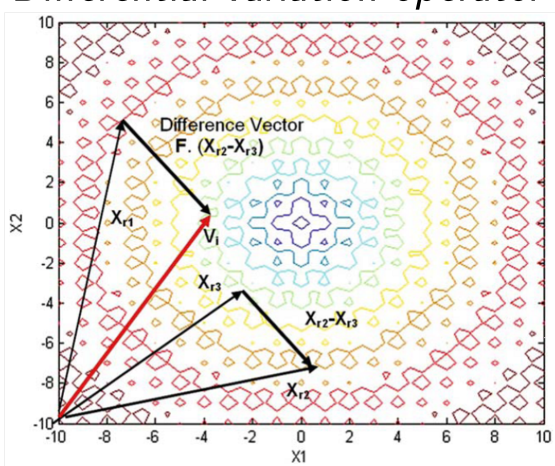
Stochastic Ranking: Bubble-sort-like Procedure

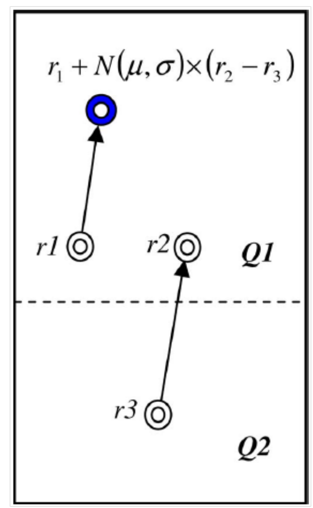
$I_j = j \ \forall\, j \in \{1, \ldots, \lambda\}$
for $i = 1$ to N **do**
    for $j = 1$ to $\lambda - 1$ **do**
        sample $u \in U(0,1)$
        if $(\phi(I_j) = \phi(I_{j+1}) = 0)$ **or** $(u < P_f)$ **then**
            if $(f(I_j) > f(I_{j+1}))$ **then**
                $swap(I_j, I_{j+1})$
            fi
        else
            if $(\phi(I_j) > \phi(I_{j+1}))$ **then**
                $swap(I_j, I_{j+1})$
            fi
        fi
    od
    if no *swap* done **break fi**

Stochastic ranking coupled to differential evolution

- Solutions (vectors) are ranked with SR before the DE operators are applied.
- The population is split into two sets based on SR
  1. **Vectors with the highest ranks** (Q_1) - from this set the **base vector**, r_1, and the vector which determines the **search direction**, r_2, are chosen at random.
  2. **Remaining vectors** (Q_2) - the other vector, r_3, is chosen at random from this set.

Differential variation operator





Stochastic Ranking: Conclusions

- SR does not use any specialized variation operators.
- SR does not require a priori knowledge about a problem since it does not use any penalty coefficient r_g in a penalty function.
- The approach is easy to implement.

Special Representation and Operators

Random keys – a representation for representing permutations, it is an efficient method for encoding ordering and scheduling problems.

- A random key vector of length l consists of l random values (keys) that are floating numbers between zero and one.

  Ex.: $\overrightarrow{r}_5 = (0.17, 0.92, 0.63, 0.75, 0.29)$
- Of importance for the interpretation of the random key vector is the position of the keys.

  The positions of the keys in the vector are ordered according to their values in ascending or descending order which gives a permutation of l numbers.

  Ex.: Random key vector $\overrightarrow{r}_5$ can be interpreted as the sequence $\overrightarrow{r}_5^s = 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 1$

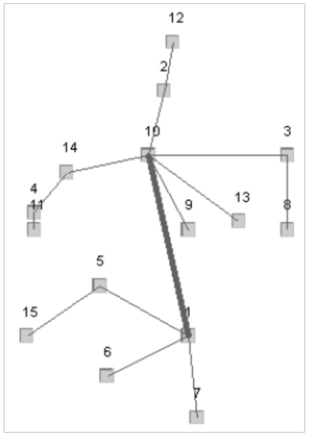
Properties of the encoding:

- A valid sequence $\overrightarrow{r}_l^s$ can always be created from the random key vector as long as there are no two keys r_i and r_j with the same values.
- There are many possibilities for the construction of each particular sequence (permutation).
- Locality of the random keys is high – a small change in the genotype (the vector $\overrightarrow{r}_l$) leads to a small change in the phenotype (the sequence $\overrightarrow{r}_l^s$).
- When using EAs with random keys, all kinds of standard crossover and mutation operators can be used that always produce only valid solutions (i.e. the interpreted permutations).

Random Keys for the Network Design Problems

Network Design Problem

- A tree network is defined as a connected graph with n nodes and $n-1$ links (there are no loops in the tree).
- Between any two nodes there exists exactly one possible path for the flow.
- The goal is to minimize the overall cost for constructing and maintaining the tree network that is calculated by summing-up the cost of all links.



Encoding tree networks with Network Random Keys (NetKeys) [Rothlauf02]

- The real-valued NetKeys are interpreted as the importance of the link. The higher the value of the allele, the higher the probability that the link is used for the tree.
- Every NetKey vector represents a valid network structure.

Constructing the tree network from the NetKey vector

1. Let G be an empty graph with n nodes, and $\overrightarrow{r}_l^s$ the sequence with length $l = n(n-1)/2$ that could be constructed from the NetKey vector $\overrightarrow{r}_l$. All possible links of G are numbered from 1 to l.

   Let $i = 0$.

2. Let j be the number at the ith position of $\overrightarrow{r}_l^s$.

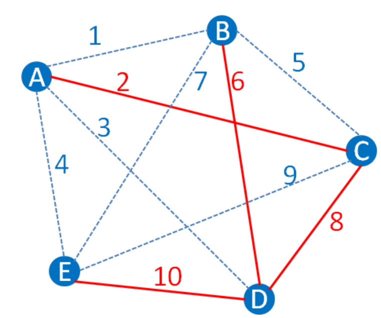
3. If the insertion of the link with number j in G would not create a cycle, then insert the link with number j in G.

4. Stop, if there are $n-1$ links in G.

5. Increment i and continue with step 2.

Ex.:

link nr.	1	2	3	4	5	6	7	8	9	10
NetKey	0.55	**0.73**	0.09	0.23	0.40	**0.82**	0.65	**0.85**	0.75	**0.90**
link	A-B	A-C	A-D	A-E	B-C	B-D	B-E	C-D	C-E	D-E

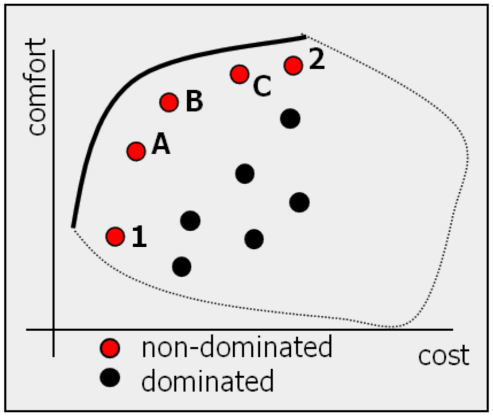


General form of multi-objective optimization problem

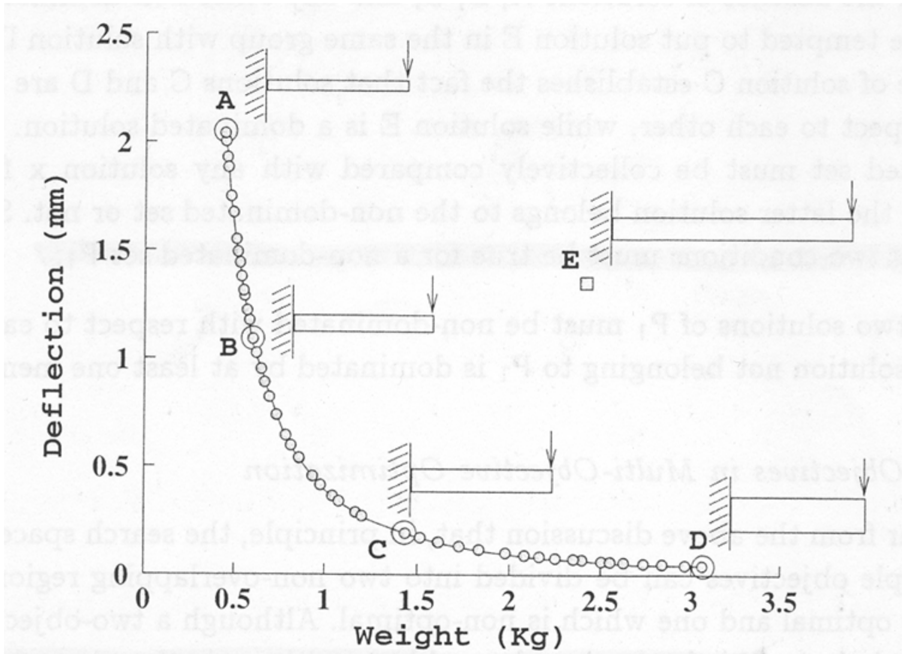
$$
\begin{array}{rll}
\text{Minimize/maximize} & f_m(x), & m = 1, 2, ..., M; \\
\text{subject to} & g_j(x) \geq 0, & j = 1, 2, ..., J; \\
& h_k(x) = 0, & k = 1, 2, ..., K; \\
& x_i^{(L)} \leq x_i \leq x_i^{(U)}, & i = 1, 2, ..., n.
\end{array}
$$

- x is a vector of n decision variables: $x = (x_1, x_2, ..., x_n)^T$;
- g_j, h_k are inequality and equality constraints, respectively.
- **Conflicting objectives**
  - A solution that is extreme with respect to one objective requires a compromise in other objectives.
  - A sacrifice in one objective is related to the gain in other objective(s).

  Motivation example: Buying a car
  - two extreme hypothetical cars 1 and 2,
  - cars with a trade-off between cost and comfort – A, B, and C.



Multiobjective Techniques: Using Pareto Schemes



Pareto dominance: *A solution $x^{(1)}$ is said to dominate the other solution $x^{(2)}$, $x^{(1)} \preceq x^{(2)}$, if $x^{(1)}$ is no worse than $x^{(2)}$ in all objectives and $x^{(1)}$ is strictly better than $x^{(2)}$ in at least one objective.*

Solutions A, B, C, D are *non-dominated* solutions (Pareto-optimal solutions)

Solution E is *dominated* by C and B (E is non-optimal).

Approaches based on Evolutionary Multiobjective Optimization

Two ways the NLP is transformed into a multiobjective optimization problem

- NLP $\longrightarrow$ **Unconstrained Bi-objective Optimization (BOP)**: Transforms the NLP into an unconstrained bi-objective optimization problem with the objectives being (1) the original objective function and (2) the sum of constraint violation.
- NLP $\longrightarrow$ **Unconstrained Multiobjective optimization (MOP)**: Transforms the NLP into an unconstrained multiobjective optimization problem where the original objective function and each constraint are treated as separate objectives.

The most popular are the MOP approaches.

Approaches based on Evolutionary Multiobjective Optimization

Two ways the NLP is transformed into a multiobjective optimization problem

- NLP $\longrightarrow$ **Unconstrained Bi-objective Optimization (BOP)**: Transforms the NLP into an unconstrained bi-objective optimization problem with the objectives being (1) the original objective function and (2) the sum of constraint violation.
- NLP $\longrightarrow$ **Unconstrained Multiobjective optimization (MOP)**: Transforms the NLP into an unconstrained multiobjective optimization problem where the original objective function and each constraint are treated as separate objectives.

The most popular are the MOP approaches.

However, multiobjective optimization does not appear to be any easier than constrained optimization since one has to balance different objectives in optimization.

Multiobjective Techniques: NPGA-based Approach

[Coello02] – Niched-Pareto Genetic Algorithm that uses **binary tournament selection** based on Pareto non-dominance.

- Parameter S_r, which indicates the minimum number of individuals that will be selected through dominance-based tournament selection.

  The remainder, $1 - S_r$, will be selected using a purely probabilistic approach.

  – **Tournament selection** – three possible situations when comparing two candidates

  1. Both are feasible. In this case, the candidate with a better fitness value wins.
  2. One is infeasible, and the other is feasible. The feasible candidate wins, regardless of its fitness function value.
  3. Both are infeasible.
     (a) Check both candidates whether they are dominated by ind. from the **comparison set**.
     (b) If one is dominated by the comparison set, and the other is not dominated then the non-dominated candidate wins.
     Otherwise, the candidate with the lowest amount of constraint violation wins, regardless of its fitness function value.

  – **Probabilistic selection** – Each candidate has a probability of 50% of being selected.

- Robust, efficient and effective approach.

Recommended Reading

[Homaifar94] Homaifar, A., Lai, S.H.Y., Qi, X.: Constrained optimization via genetic algorithms, Simulation 62 (4), pp. 242–254, 1994.

[Joines94] Joines, J., Houck, C.: On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs, in: D. Fogel (Ed.), Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, Orlando, FL, pp. 579–584, 1994.

[Runarsson00] Runarsson, T. P. and Yao, X.: Stochastic Ranking for Constrained Evolutionary Optimization, IEEE Transactions on Evolutionary Computation, 4(3):284–294, September 2000.

[Farmani03] Farmani, R. and Wright, J. A.: Self-Adaptive Fitness Formulation for Constrained Optimization, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 7, NO. 5, 2003.

[Hamida00] Hamida, S.B. and Schoenauer, M.: An Adaptive Algorithm for Constrained Optimization Problems, Parallel Problem Solving from Nature PPSN VI, 2000, pp. 529-538, 2000.

Recommended Reading

[Zhou03] Zhou, Y., Li, Y., He, J., Kang, L.: Multi-objective and MGG Evolutionary Algorithm for Constrained Optimization. In: Proceedings of the Congress on Evolutionary Computation 2003 (CEC'2003). Volume 1., Piscataway, New Jersey, Canberra, Australia, IEEE Service Center (2003) 1–5.

[Hadj-Alouane92] Hadj-Alouane, A.B., Bean, J.C.: A Genetic ALGORITHM FOR THE Multiple-choice Integer Program. Technical Report TR 92-50, Department of Industrial and Operations Engineering, The University of Michigan, 1992.

[Rothlauf02] Rothlauf, F., Goldberg, D.E., and Heinzl, A.: Network random keys — A tree network representation scheme for genetic and evolutionary algorithms, Evolutionary Computation, vol. 10, no. 1, pp.75 - 97 , 2002.

[Venkatraman05] Venkatraman, S., Yen, G.G.: A Generic Framework for Constrained Optimization Using Genetic Algorithms. IEEE Transactions on Evolutionary Computation 9(4) (2005).

Recommended Reading

[Coello00] Coello, C.A.: Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. Engineering Optimization 32(3) (2000) 275–308.

[Coello02] Coello, C.A.C., Mezura-Montes, E.: Handling Constraints in Genetic Algorithms Using Dominance-Based Tournaments. In Parmee, I., ed.: Proceedings of ACDM'2002. Volume 5., University of Exeter, Devon, UK, Springer-Verlag (2002) 273–284.