A0M33EOA
Constraint Handling in Evolutionary Algorithms

Petr Pošík

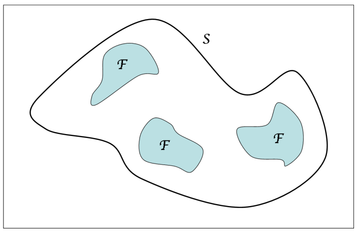
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Heavilly using slides from **Jiří Kubalík**, CIIRC CTU, with permission.

Definition

A **constrained optimization problem** is defined as

minimize $f(x)$
subject to $x \in \mathcal{F} \subset \mathcal{S}$,



where

- $\mathcal{S}$ is a space of all possible solutions with a chosen representation, and
- $\mathcal{F}$ is its **subspace of feasible solutions**, usually defined by a set of **constraints** the solution must fulfill.

Various kinds of constrained problems arise depending on

- whether the value of objective function $f(x)$ can be computed even for infeasible individuals,
- whether the constraints return
  - only a boolean info (fulfilled/unfulfilled, feasible/infeasible), or
  - a more informative degree of constraint violation,
- whether the constraints are black-box or not, ...

In this lecture we shall focus on nonlinear programming problems where

- $f(x)$ can be evaluated for $x \notin \mathcal{F}$ and
- degree of constraint violation is available.

Nonlinear Programming Problem

The general nonlinear programming problem (NLP) can be formulated as follows:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}), \quad \mathbf{x} = (x_1, \ldots, x_D), \\ &\text{subject to } g_i(\mathbf{x}) \le 0, \quad i = 1, 2, ..., m, \\ &\qquad\qquad h_j(\mathbf{x}) = 0, \quad j = 1, 2, ..., p, \end{aligned}$$

where

- $\mathbf{x}$ is a vector of D decision variables,
- each x_d, $d = 1, \ldots, D$ is bounded by lower and upper limits $x_d^{(L)} \le x_d \le x_d^{(U)}$, 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:

$$|h_j(\mathbf{x})| - \varepsilon \le 0$$

where ε is the tolerance allowed.

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).

Presented Constraint-Handling Approaches

- Simple penalty functions
- Adaptive penalty and Stochastic Ranking
- Special representations and operators
- Multi-objective optimization techniques

Penalization

Penalizing Infeasible Solutions

Penalization transforms a constrained problem

$$\begin{aligned}&\text{minimize } f(x)\\&\text{subject to } x \in \mathcal{F},\end{aligned}$$

into an unconstrained problem

$$\text{minimize } f_P(x) = f(x) + p(x),$$

where the **penalty**

$$p(x) \begin{cases} = 0 & \text{if } x \in \mathcal{F}, \\ > 0 & \text{otherwise.} \end{cases}$$

Individual penalty methods differ in the design of penalty function $p(x)$.

There are 4 basic categories of penalty functions based on the way their parameters are determined:

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

Death Penalty

Death Penalty: reject infeasible individuals.

- If a solution violates any constraint, it is rejected and generated again.
- No info about the degree of infeasibility of such a solution is needed.
- The easiest and computationally efficient way to handle constraints.

Criticism:

- No exploitation of the information from infeasible solutions.
- The search may ”stagnate” in the presence of very small feasible regions.
- Useable only for problems where the proportion of feasible region to the whole search space is fairly large.
- A variation that assigns a *very bad fitness*, i.e., a *very large penalty*, to infeasible solutions may work better in practice.

Penalty Based on the Number of Unsatisfied Constraints

Fitness of an individual is determined using:

$$f_P(x) = \begin{cases} f(x) & \text{if the solution is feasible,} \\ K(1 + \frac{u}{m+p}) & \text{otherwise.} \end{cases}$$

where

- u is the number of unsatisfied constraints, and
- K is a sufficiently large constant.

If an individual x is infeasible:

- $f_P(x) \in (K, 2K\rangle$,
- $f_P(x)$ is always worse than a fitness of any other feasible individual (if K is sufficiently large), and
- $f_P(x)$ is the same as the fitness of all the individuals that violate the same number of constraints.

Penalty based on Constraint Violation

Penalty based on constraint violations

- **transform a constrained optimization problem into an unconstrained one**
- by adding the amount of constraint violation present in the solution to the objective function value:

$$f_P(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m+p} r_i v_i(\mathbf{x}),$$

where

- $v_i(\mathbf{x})$ are functions of the **constraint violation** (depending on $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$), and
- r_i are positive constants called **penalty coefficients** or penalty factors.

A common form of constraint violation functions v_i

- for inequality constraints g_i, $i \in \{1, \ldots, m\}$,

$$v_i(\mathbf{x}) = \max(0, g_i(\mathbf{x})),$$

- for equality constraints h_i, $i \in \{1, \ldots, p\}$,

$$v_{m+i}(\mathbf{x}) = |h_i(\mathbf{x})|$$

or

$$v_{m+i}(\mathbf{x}) = \max(0, |h_i(\mathbf{x})| - \varepsilon).$$

Static Penalty

Static penalty: the penalty coefficients r_i in penalized fitness

$$f_P(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m+p} r_i v_i(\mathbf{x})$$

do not depend on the current generation number, they remain constant during the entire evolution.

- r_i can itself be an (increasing) function of constraint violation, $r_i(v_i(\mathbf{x}))$.
- E.g., in [HQL94], $r_i(v_i(\mathbf{x}))$ was an increasing stepwise function depending on a level of constraint violation.

Criticism:

- What function of $v_i(\mathbf{x})$ to choose? How to set up its parameters? Very often we have to specify a lot of parameters, which heavilly affect the results.
- Penalty coefficients are difficult to generalize. They are problem-dependent.
- Keeping the same penalty coefficient along the entire evolution is not a good idea.
  The population evolves; why should the coefficients directing the search be static?

[HQL94] Abdollah Homaifar, Charlene X. Qi, and Steven H. Lai. Constrained optimization via genetic algorithms. *SIMULATION*, 62(4):242–253, 1994.

Dynamic Penalty

Dynamic Penalty: the **current generation number influences the penalty coefficients**.

Typically, the penalty coefficients **increase over time**, pushing the search towards the feasible region.

The approach from [JH94] evaluates individuals as follows:

$$f_P(\mathbf{x}) = f(\mathbf{x}) + (Ct)^\alpha \cdot V(\mathbf{x})$$

where

- t is the generation number,
- C and α are user-defined constants (recommended values are $C = 0.5$ or $C = 0.05$, $\alpha = 1$ or $\alpha = 2$),
- $V(\mathbf{x})$ is the **sum of constraint violations** and is defined as:

  $$V(\mathbf{x}) = \sum_{i=1}^{m+p} v_i(\mathbf{x})$$

  where $v_i(\mathbf{x})$ are the constraints violation functions (containing $g_i(x)$ 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.

[JH94] Jeffrey A. Joines and Christopher R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In *Proc. of the 1st IEEE Conference on Evolutionary Computation*, pages 579–584. IEEE Press, 1994.

Dynamic Penalty: Criticism

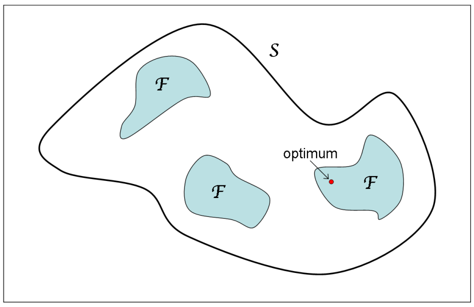
- It is **difficult to derive good schedules for dynamic penalty functions** in practice.
- Dynamic penalties are **sensitive to values of** α **and** C and there are no guidelines for choosing proper values for a particular problem.
- If a bad penalty coefficient is chosen, EA may find
  - non-optimal feasible solutions (if the penalty is too high), or
  - infeasible solutions (if the penalty is too low).
- No constraint normalization technique is used, thus **certain constraint may undesirably dominate** the whole penalty value.

Motivation

Assume the penalty fitness function of the following form:

$$f_P(\boldsymbol{x}) = f(\boldsymbol{x}) + r_g V(\boldsymbol{x})$$

Setting an optimal (or near-optimal) value of 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 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

Adaptive penalty uses feedback from the current state of the search process:

$$f_P(\boldsymbol{x}) = f(\boldsymbol{x}) + r_g(t) V(\boldsymbol{x})$$

- $r_g(t)$ is updated every generation.
- It can increase or decrease according the quality of the solutions in current population.

Here we present the following selection of adaptive penalty approaches:

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

GA with Non-linear Penalty Function

[BHAB97] used a penalized fitness

$$f_P(\mathbf{x}) = f(\mathbf{x}) + r_g(t) \sum_{i=1}^{m+p} v_i(x)^2$$

and the penalty factor r_g was updated according to the following rule:

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

where

- case #1: the best individual in all the last k generations was **always feasible**,
- case #2: the best individual in all the last k generations was **always infeasible**,
- $\beta_1 > 1, \beta_2 > 1$, and $\beta_1 \neq \beta_2$ (to avoid cycling). E.g., $\beta_1 = 2.8$, $\beta_2 = 4$.

Intuition:

- The penalty is decreased if the algorithm can sample feasible solutions.
- The penalty is increased if the algorithm cannot sample feasible solutions.
- It tries to avoid having either an all-feasible or an all-infeasible population.
- The problem is how to choose a proper time window (k) and the values of β_1 and β_2.

[BHAB97] Atidel Ben Hadj-Alouane and James C. Bean. A genetic algorithm for the multiple-choice integer program. *Oper. Res.*, 45(1):92–101, February 1997.

Adaptive Segregational Constraint Handling EA (ASCHEA)

ASCHEA [BHS00] maintains both feasible and infeasible individuals in the population (when necessary).

Adaptive mechanisms based on three main components:

1. **Adaptive penalty function** updates the penalty coefficients according to the **proportion of feasible individuals** in the current population.
2. **Constraint-driven mate selection** **mates feasible individuals with infeasible ones** and thus explores the region around the boundary of the feasible domain.
3. **Segregational replacement strategy** **favors** a given number of **feasible individuals** in the population.

[BHS00] Sana Ben Hamida and Marc Schoenauer. An Adaptive Algorithm for constrained optimization problems. In *PPSN 2000*, Paris, France, September 2000.

ASCHEA: Adaptive Penalty

ASCHEA uses penalty function of the following form:

$$p(\boldsymbol{x}) = r_g(t)V(\boldsymbol{x})$$

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

$$r_g(t+1) = \begin{cases} \frac{1}{c} \cdot r_g(t) & \text{if } \tau_t > \tau_{target}, \\ c \cdot r_g(t) & \text{otherwise.} \end{cases}$$

where

- $c > 1$ is a user-defined parameter, a recommended value is around 1.1;
- a recommended value of τ_{target} is around 0.6.

In a newer version of ASCHEA, the authors used a separate penalty factor for each constraint

$$p(\boldsymbol{x}) = \sum_{i=1}^{m+p} r_i(t)v_i(\boldsymbol{x})$$

and updated each $r_i(t)$ separately based on the proportion of individuals in the population that fulfill that particular constraint.

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 the first parent x_1:

- if $\tau_t < \tau_{target}$ and x_1 is feasible, select x_2 among infeasible solutions only,
- otherwise, select x_2 according to fitness.

ASCHEA: Segregational Replacement

Segregational replacement: Deterministic replacement mechanism used in ES-like fashion 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 old parents in case of the $(\mu + \lambda)$-ES

are selected to the new population in the following way:

1. **Feasible solutions are selected without replacement** based on their fitness, until $\tau_{select}\mu$ have been selected, or no more feasible solutions are available.
2. The rest of population is then filled with **the remaining individuals**, based on the penalized fitness.

τ_{select} is a user-defined proportion of feasible solutions which are considered superior to all infeasible solutions. (Recommendation: $\tau_{select} \approx 0.3$.)

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.

What Do Penalty Methods Do?

Assume the penalized fitness function of the following form:

$$f_P(\mathbf{x}) = f(\mathbf{x}) + r_g V(\mathbf{x}).$$

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

$$f_p(\mathbf{x}_1) \le f_P(\mathbf{x}_2) \le \cdots \le f_P(\mathbf{x}_\lambda). \quad (1)$$

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

$$f_i + r_g V_i \le f_{i+1} + r_g V_{i+1}, \qquad \text{where } f_i = f(\mathbf{x}_i) \text{ and } G_i = G(\mathbf{x}_i), \quad (2)$$

we define so called **critical penalty coefficient**

$$\check{r}_i = (f_{i+1} - f_i)/(V_i - V_{i+1}) \qquad \text{for } V_i \ne V_{i+1}. \quad (3)$$

What Do Penalty Methods Do? (Cont.)

For a given $r_g > 0$, there are three different cases for which the inequality (2) holds:

1. $f_i < f_{i+1}$ and $V_i \ge V_{i+1}$: **Objective function is dominant** in determining the inequality and the value of r_g must fulfill $0 < r_g < \check{r}_i$.

Example:

$f_i = 10, \quad V_i = 7$
$f_{i+1} = 20 \quad V_{i+1} = 5$
$\check{r}_i = (20-10)/(7-5) = 5 \quad \Longrightarrow \quad 0 < r_g < 5$

$r_g = 4:$	$38 \le 40$	inequality (2) holds
$r_g = 6:$	$52 \not\le 50$	inequality (2) does not hold

2. $f_i \ge f_{i+1}$ and $V_i < V_{i+1}$: **Penalty function is dominant** in determining the inequality and the value of r_g should be $\check{r}_i < r_g$.

Example:

$f_i = 20, \quad V_i = 5$
$f_{i+1} = 10 \quad V_{i+1} = 7$
$\check{r}_i = (10-20)/(5-7) = 5 \quad \Longrightarrow \quad 5 < r_g$

$r_g = 4:$	$40 \not\le 38$	inequality (2) does not hold
$r_g = 6:$	$50 \le 52$	inequality (2) holds

3. $f_i < f_{i+1}$ and $V_i < V_{i+1}$: The inequality is satisfied for any $r_g > 0$, $\check{r}_i < 0$. Neither the objective nor the penalty function are dominant.

Example:

$f_i = 10, \quad V_i = 5$
$f_{i+1} = 20 \quad V_{i+1} = 7$
$\check{r}_i = (20-10)/(5-7) = -5 \quad \Longrightarrow \quad$ inequality (2) holds for all $r_g > 0$

When comparing feasible solutions, or solutions where one dominates the other, r_g has no impact on inequality (2).

Value of r_g must be in range $\underline{r}_g < r_g < \overline{r}_g$ to be able to influence the ranking of individuals:

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

What Do 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. $\underline{r}_g < r_g < \overline{r}_g$: Comparisons use a combination of objective and penalty values.

   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 of search.

Stochastic Ranking

Stochastic Ranking [RY00] characteristics:

- Easy to implement.
- The resulting ranks can be used with any other rank-based selection technique (truncation, tournament, . . .), and thus easily incorporated in any EA.
- **The balance between objective and penalties** is achieved directly and explicitly.
- It does not need the penalty coefficient r_g.

  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.
- When comparing two solutions x_i and x_j:
  - if both solutions are feasible, compare their objective values $f(x_i)$ and $f(x_j)$;
  - if at least one solution is infeasible,
    - with probability P_f compare their objective values $f(x_i)$ and $f(x_j)$,
    - with probability $1 - P_f$ compare their penalty values $G(x_i)$ and $G(x_j)$.
- Recommended range of P_f values is $(0.4, 0.5)$
- The P_f introduces a stochastic component to the ranking process, so that **some solutions may get a good rank even if they are infeasible**.

[RY00] T.P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

Stochastic Ranking: Bubble-sort-like Procedure

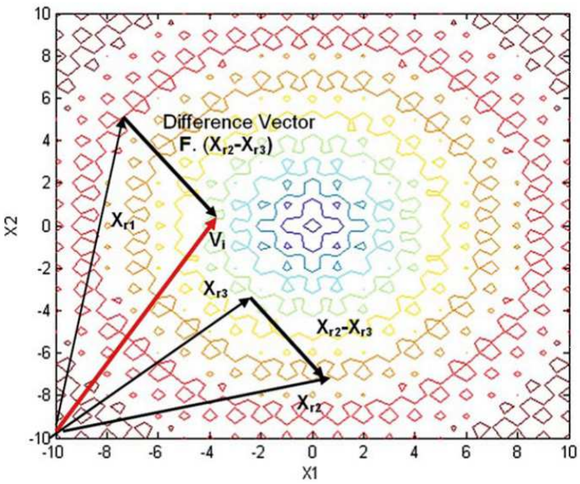
```
I_j = j ∀ j ∈ {1, ... , λ}
for i = 1 to N do
    for j = 1 to λ − 1 do
        sample u ∈ U(0, 1)
        if (φ(I_j) = φ(I_{j+1}) = 0) or (u < P_f) then
            if (f(I_j) > f(I_{j+1})) then
                swap(I_j, I_{j+1})
            fi
        else
            if (φ(I_j) > φ(I_{j+1})) then
                swap(I_j, I_{j+1})
            fi
        fi
    od
    if no swap done break fi
```

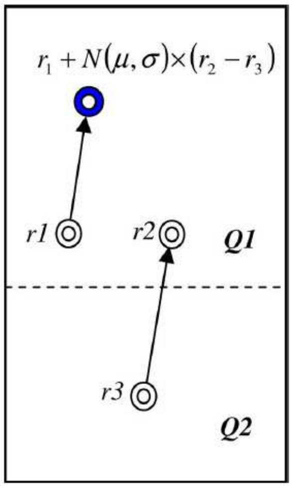

- I_j is the jth individual, λ is population size
- N is the number of passes throught the population
- $\phi(I_j)$ is the sum of constraint violations of individual I_j
- $f(I_j)$ is the (non-penalized) fitness of individual I_j

Stochastic Ranking + DE

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.





Stochastic Ranking: Conclusions

- SR does not use any specialized variation operators.
- SR does not require prior 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 Representations and Operators

Random Keys

Random keys:

- Genotype/phenotype mapping for permutations. Suitable, e.g., for ordering and scheduling problems.
- A random key vector (a genotype) of length l consists of l floating-point values (keys), usually between 0 and 1.

$$\text{Example: } \boldsymbol{r}_5 = (0.17, 0.92, 0.63, 0.75, 0.29)$$

- Such vector is interpreted as permutation (phenotype) using the ranks of individual items.

$$\text{Example: Random key vector } \boldsymbol{r}_5 \text{ can be interpreted as permutation } \boldsymbol{r}_5^s = (1, 5, 3, 4, 2)$$

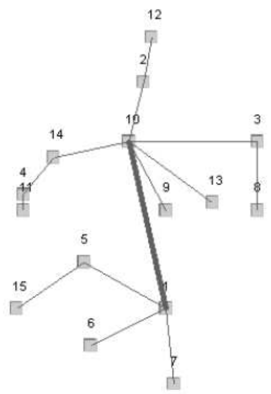
Properties of the encoding:

- A valid permutation $\boldsymbol{r}_l^s$ can always be created from any random key vector, if all keys are unique.
- There are many random key vectors that are mapped to the same sequence (permutation).
- Locality of the random keys is high: a small change in the genotype (the vector $\boldsymbol{r}_l$) leads to a small change in the phenotype (the sequence $\boldsymbol{r}_l^s$).
- EAs with random keys can use all kinds of standard crossover and mutation operators (they will always produce only valid 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. (It is a spanning tree of a graph.)
- There is only one possible path between any two nodes.
- The goal is to minimize the overall cost for constructing and maintaining the tree network, calculated as a sum of the costs of all links. (Find minimum spanning tree.)



Remarks:

- There are deterministic algorithm for finding the optimal spanning tree (Prim, Kruskal, ...). These will always return the same optimal solution, given the evaluation of individual links.
- We still may want to approach the problem with a heuristic algorithm, e.g.,
  - when we want to find near-optimal alternative solutions,
  - when we have more objectives and want to provide Pareto-optimal solutions, ...

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.

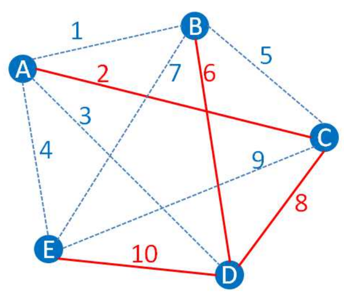
Random Keys for the Network Design Problems (cont.)

Constructing the tree network from a NetKey vector (similar to Kruskal MST algorithm)

1. Let G be an empty graph with n nodes, and $\boldsymbol{r}_l^s$ the sequence with length $l = n(n-1)/2$ that constructed from NetKey vector $\boldsymbol{r}_l$. 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 $\boldsymbol{r}_l^s$.
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.

Example:

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



Approaches based on Evolutionary Multiobjective Optimization

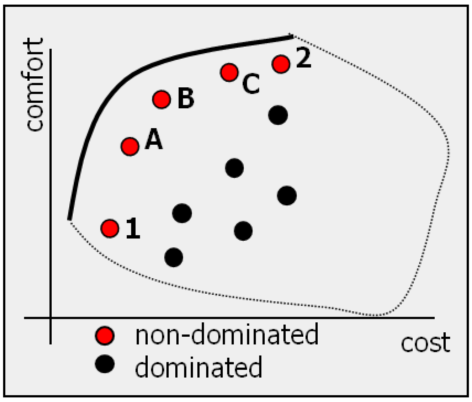
General form of multi-objective optimization problem

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

- x is a vector of n decision variables: $x = (x_1, x_2, \ldots, x_n)$;
- 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).
- The goal is to find multiple trade-off solutions (well spread and close to Pareto front).



EMO-based Constraint Handling

Two options to turn the NLP 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 Multi-objective optimization (MOP)**: Transforms the NLP into an unconstrained multiobjective optimization problem with the objectives being
  1. the original objective function,
  2. violation of constraint 1,
  3. violation of constraint 2,
  4. ...

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

Niched-Pareto Genetic Algorithm [Coello02] 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.

Summary

Learning outcomes

After this lecture, a student shall be able to

- define the general nonlinear programming problem with constraints;
- define the general form of fitness function with penalty function;
- describe the main categories of constraint handling approaches with penalty function – death penalty, static penalty, dynamic penalty, adaptive penalty – and explain their shortcomings;
- describe the GA with non-linear penalty function and its main characteristics;
- list and describe the principal components of the Adaptive Segregational Constraint Handling EA (ASCHEA);
- explain the main idea behind the Stochastic Ranking adaptive penalty approach;
- implement random keys representation used for solving permutation problems;
- explain the main idea behind multi-objective approaches to constraint handling.