# Constraints Handling in Optimization with Metaheuristic Algorithms (support for Lecture 12)

Daniela Zaharie

# Motivation

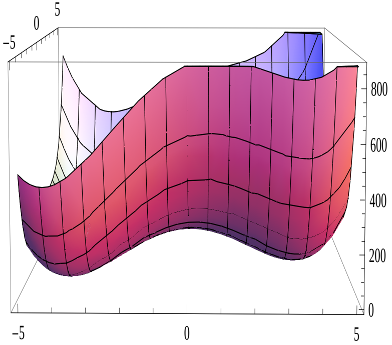
- Most of real world optimization problems are constrained
- Types of constraints
  - Bound constraints: $a_j \leq x_j \leq b_j$ for $j = \overline{1, n}$
  - Inequality constraints: $g_i(x) \leq 0$ for $i = \overline{1, p}$
  - Equality constraints: $h_i(x) = 0$ for $i = \overline{1, q}$ (usually transformed in $|h_i(x)| \leq \epsilon$)

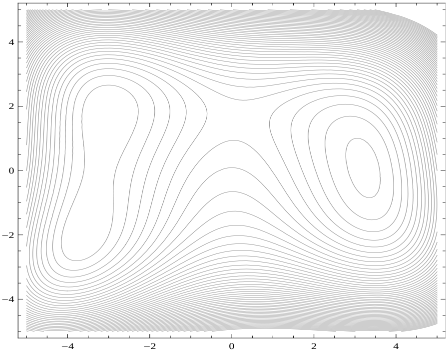
# Motivation

- Most of real world optimization problems are constrained
- Types of constraints
  - Bound constraints: $a_j \leq x_j \leq b_j$ for $j = \overline{1, n}$
  - Inequality constraints: $g_i(x) \leq 0$ for $i = \overline{1, p}$
  - Equality constraints: $h_i(x) = 0$ for $i = \overline{1, q}$ (usually transformed in $|h_i(x)| \leq \epsilon$)

# Motivation

A simple example: $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2)^2$, $x_1, x_2 \in [-5, 5]$

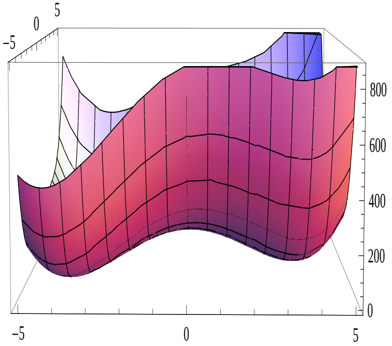


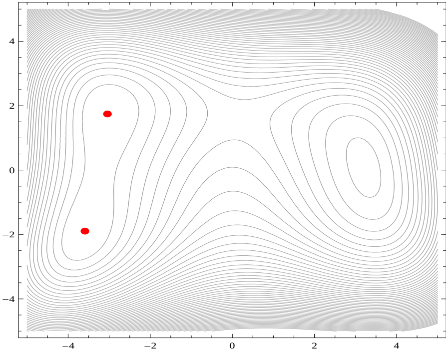




- Where is (are) the optimum (optima)?

# Motivation

A simple example: $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2)^2$, $x_1, x_2 \in [-5, 5]$

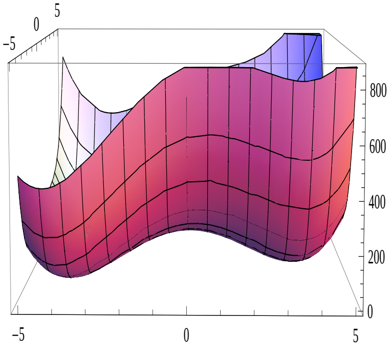


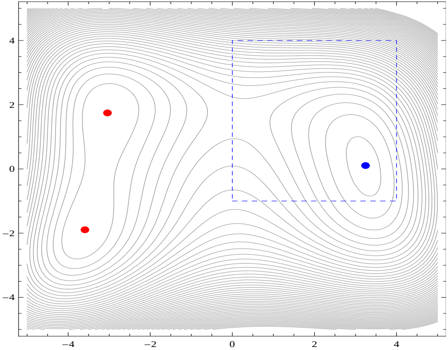




- What about the case when the feasible region is smaller (e.g. $[0, 4] \times [-1, 4]$ instead of $[-5, 5] \times [-5, 5]$)?

# Motivation

A simple example: $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2)^2$, $x_1, x_2 \in [-5, 5]$




- Global unfeasible optima (red points): $f(-3.59, -1.89) = 0.00032$, $f(-3.04, 1.74) = 0.00049$
- Feasible optimum (blue point): $f(3.29, 0.08) = 10.87$
- The search should be directed toward the feasible region defined by the bound constraints

# Outline

- Overview of constraint handling methods
  - penalty functions
  - feasibility rules
  - stochastic ranking
  - $\epsilon$-constraints
- Particular methods for handling bound constraints
  - resampling
  - random reinitialization
  - projection
  - reflection

# Overview of constraint handling methods

## Constrained optimization problems

find $x$ which minimizes $f(x)$ subject to

- $a_j \leq x_j \leq b_j$ (bound constraints)
- $g_i(x) \leq 0,\ i = \overline{1,p}$ (inequality constraints)
- $h_i(x) = 0,\ i = \overline{1,q}$ (equality constraints)

Main approaches:

- Search only the feasible region (e.g. start with a feasible element and keep the constraints satisfied)
  - rather easy for bound constraints
  - for general constraints it might be difficult even to find initial feasible positions
- Allow the search outside the feasible region but favor the feasible or almost feasible elements
  - Question: How can be decided that an element is almost feasible?
  - Answer: By estimating the amount of violated constraints

# Overview of constraint handling methods

## Constrained optimization problems

find $x$ which minimizes $f(x)$ subject to

- $a_j \leq x_j \leq b_j$ (bound constraints)
- $g_i(x) \leq 0,\ i = \overline{1,p}$ (inequality constraints)
- $h_i(x) = 0,\ i = \overline{1,q}$ (equality constraints)

Main approaches:

- Search only the feasible region (e.g. start with a feasible element and keep the constraints satisfied)
  - rather easy for bound constraints
  - for general constraints it might be difficult even to find initial feasible positions
- Allow the search outside the feasible region but favor the feasible or almost feasible elements
  - Question: How can be decided that an element is almost feasible?
  - Answer: By estimating the amount of violated constraints

# Quantifying the constraint violation

- Number of violated constraints
  - does not express the distance to the feasible region
- Amount of violation

$$\phi(x) = \sum_{i=1}^{p} \max\{0, g_i(x)\} + \sum_{i=1}^{q} |h_i(x)|$$

  - $\phi(x) = 0$ means that the constraints are satisfied
  - smaller values of $\phi(x)$ correspond to elements "closer" to the feasible region
  - can be interpreted as a second optimization criterion which can be used to influence the selection (ranking) of the elements $\Longrightarrow$ bias the search toward the feasible region

# Penalty functions method

## Main idea

Penalize the infeasible solutions by increasing the value of the objective function based on the amount of constraint violation

## Implementation

New objective function

$$F(x) = f(x) + \sum_{i=1}^{p} \alpha_i \cdot \max\{0, g_i(x)\} + \sum_{i=1}^{q} \beta_i \cdot |h_i(x)|$$

## Advantages

- easy to implement

## Disadvantages

- sensitive to the values of the penalty factors ($\alpha_i$, $\beta_i$) which are problem-dependent

# Penalty functions method

## Main idea

Penalize the infeasible solutions by increasing the value of the objective function based on the amount of constraint violation

## Implementation

New objective function

$$F(x) = f(x) + \sum_{i=1}^{p} \alpha_i \cdot \max\{0, g_i(x)\} + \sum_{i=1}^{q} \beta_i \cdot |h_i(x)|$$

## Advantages

- easy to implement

## Disadvantages

- sensitive to the values of the penalty factors ($\alpha_i$, $\beta_i$) which are problem-dependent

# Penalty functions method

## Main idea

Penalize the infeasible solutions by increasing the value of the objective function based on the amount of constraint violation

## Implementation

New objective function

$$F(x) = f(x) + \sum_{i=1}^{p} \alpha_i \cdot \max\{0, g_i(x)\} + \sum_{i=1}^{q} \beta_i \cdot |h_i(x)|$$

## Advantages

- easy to implement

## Disadvantages

- sensitive to the values of the penalty factors ($\alpha_i$, $\beta_i$) which are problem-dependent

# Penalty functions method

## Main idea

Penalize the infeasible solutions by increasing the value of the objective function based on the amount of constraint violation

## Implementation

New objective function

$$F(x) = f(x) + \sum_{i=1}^{p} \alpha_i \cdot \max\{0, g_i(x)\} + \sum_{i=1}^{q} \beta_i \cdot |h_i(x)|$$

## Advantages

- easy to implement

## Disadvantages

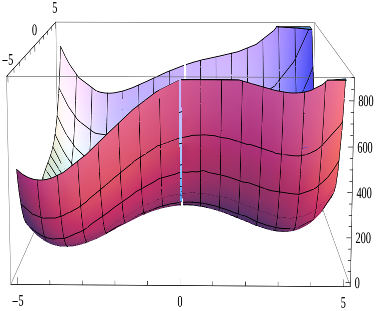
- sensitive to the values of the penalty factors ($\alpha_i$, $\beta_i$) which are problem-dependent

# Penalty functions method

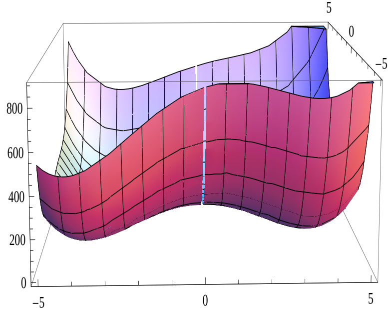
## A simple example

Constraint:

$$x_1 > 0 \Longrightarrow -x_1 \leq 0 \Longrightarrow F(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2)^2 + \alpha \cdot \max\{0, -x_1\}$$




$\alpha = 1$



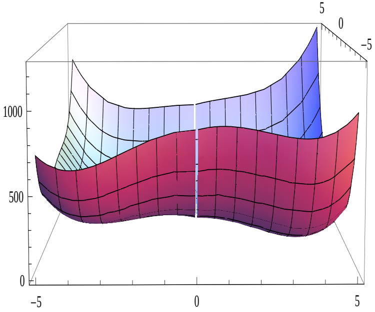

$\alpha = 10$

# Penalty functions method

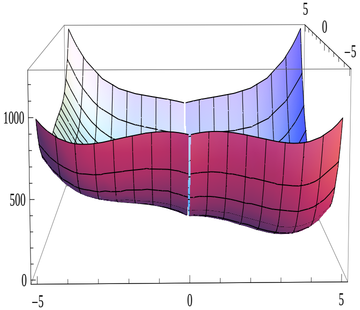
## A simple example

Constraint:

$$x_1 > 0 \Longrightarrow -x_1 \leq 0 \Longrightarrow F(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2)^2 + \alpha \cdot \max\{0, -x_1\}$$




$\alpha = 50$




$\alpha = 100$

# Feasibility rules

Deb's approach

## Main idea

use separate objective value ($f$) and penalty value $=$ degree of constraint violation ($\phi$) when compare two elements [a]

[a] K. Deb, An Efficient Constraint Handling Method for Genetic Algorithms, 2000

## Implementation (for a minimization problem)

$x$ is better than $x'$ if:

- $x$ and $x'$ are both feasible and $f(x) < f(x')$
- $x$ is feasible and $x'$ is not feasible
- $x$ and $x'$ are both unfeasible and $\phi(x) < \phi(x')$

# Feasibility rules

Deb's approach

## Advantages

- easy to implement and to couple with various search algorithms
- it does not require parameters

## Disadvantages

- separating the constraints and the objective function can lead to diversity loss (because the approach strongly favor the feasible solutions)
  - Solution: use diversity enhancement mechanisms (e.g. random elements)
- combining the constraint violations in one function ($\phi(x)$) might lead to losing the particularities of each of the constraints
  - Solution: use a Pareto ranking approach over the constraint violation values computed separately per constraint

# Stochastic ranking

## Main idea

- decides randomly which selection criterion to use (objective or penalty function)
- in some cases (random decision) two solutions are compared based only on the objective function, even if they are not both of them feasible

## Implementation

$x$ is better than $x'$ if

$$\begin{cases} ((\phi(x) = \phi(x') = 0) \text{ or } (\mathrm{rand}(0,1) < P_f)) \text{ and } (f(x) < f(x')) \\ \phi(x) < \phi(x') \end{cases}$$

# Stochastic ranking

## Advantages

- it limits the diversity loss (by accepting promising but unfeasible candidates)

## Disadvantages

- it requires the specification of a parameter ($P_f$) - the algorithm behaviour might be sensitive to the value of $P_f$ (a value used in papers: $P_f = 0.45$ [a])

[a] T.Runarsson, X. Yao- Stochastic Ranking for Constrained Evolutionary Optimization, IEEE TEvC, 2000

# $\epsilon$-Constrained Methods

## Main idea

- if both elements are feasible, slightly infeasible or have the same amount of constraint violation, they are compared based on the objective function
- if both elements are infeasible, they are compared based on their amount of constraint violation.

## Implementation

$x$ is better than $x'$ if

$$\begin{cases} f(x) < f(x') & \text{in the case when } \phi(x) \leq \epsilon, \phi(x') \leq \epsilon \\ f(x) < f(x') & \text{in the case when } \phi(x) = \phi(x') \\ \phi(x) < \phi(x') & \text{otherwise} \end{cases}$$

# $\epsilon$-Constrained Methods

## Advantages

The ranking process can be controlled by $\epsilon$

- $\epsilon = \infty$ - only the objective function is used
- $\epsilon = 0$ - lexicographic order (constraint violation first, then the objective function)

## Disadvantages

Sensitive to the value of $\epsilon$

# Bound constraints handling

- Bound constraints: $a_j \leq x_j \leq b_j$
- Aim: repair the infeasible elements ($x_j < a_j$ or $x_j > b_j$ for at least one component $j$)
- Characteristics of the repairing method to be analyzed:
  - Does it preserve some information from the infeasible element?
  - Does it preserve the characteristics of the search process or it introduces a bias (e.g. by favoring only some subregions of the feasible region)?

Variants: resampling, random reinitialization, projection, reflection

## Main idea

- Ignore the infeasible element and generate a new one by selecting new parents or other values of some control parameters
- The resampling can be done at the level of components or at the level of the full vector

## Advantages

- Easy implementation(repeated generation of new elements until a feasible one is obtained)
- It preserves the characteristics of the search strategy (no specific bias)

## Disadvantages

- The repeated generation of new candidates might be costly especially in the case when the full vector is recosntructed

## Main idea

- Ignore the infeasible element and generate a new one by selecting new parents or other values of some control parameters
- The resampling can be done at the level of components or at the level of the full vector

## Advantages

- Easy implementation(repeated generation of new elements until a feasible one is obtained)
- It preserves the characteristics of the search strategy (no specific bias)

## Disadvantages

- The repeated generation of new candidates might be costly especially in the case when the full vector is recosntructed

# Bound constraints handling

Resampling

## Main idea

- Ignore the infeasible element and generate a new one by selecting new parents or other values of some control parameters
- The resampling can be done at the level of components or at the level of the full vector

## Advantages

- Easy implementation(repeated generation of new elements until a feasible one is obtained)
- It preserves the characteristics of the search strategy (no specific bias)

## Disadvantages

- The repeated generation of new candidates might be costly especially in the case when the full vector is recosntructed

# Bound constraints handling

Random reinitialization

## Main idea

- The components which violate the constraints are randomly reinitialized in the bounding box

$$\text{if } x_j < a_j \text{ or } x_j > b_j \text{ then } x_j = random(a_j, b_j)$$

- It looses the previous search direction (at least for reinitialized components)

## Advantages

- Easy implementation and small costs
- If it is based on an uniform distribution then it does not introduce any specific bias
- It increases the population diversity (helps in avoiding premature convergence)

## Disadvantages

- It might slow down the convergence

# Bound constraints handling

Random reinitialization

## Main idea

- The components which violate the constraints are randomly reinitialized in the bounding box

$$\text{if } x_j < a_j \text{ or } x_j > b_j \text{ then } x_j = random(a_j, b_j)$$

- It looses the previous search direction (at least for reinitialized components)

## Advantages

- Easy implementation and small costs
- If it is based on an uniform distribution then it does not introduce any specific bias
- It increases the population diversity (helps in avoiding premature convergence)

## Disadvantages

- It might slow down the convergence

# Bound constraints handling

Random reinitialization

## Main idea

- The components which violate the constraints are randomly reinitialized in the bounding box

$$\text{if } x_j < a_j \text{ or } x_j > b_j \text{ then } x_j = random(a_j, b_j)$$

- It looses the previous search direction (at least for reinitialized components)

## Advantages

- Easy implementation and small costs
- If it is based on an uniform distribution then it does not introduce any specific bias
- It increases the population diversity (helps in avoiding premature convergence)

## Disadvantages

- It might slow down the convergence

# Bound constraints handling

Projection

## Main idea

- The components which violate the constraints are replaced with the closest bound

$$x_j = \begin{cases} a_j & \text{if } x_j < a_j \\ b_j & \text{if } x_j > b_j \end{cases}$$

- It preserves the previous search direction

## Advantages

- Easy implementation and small costs
- Useful when the optimum is on the bounds

## Disadvantages

- It introduces a bias in the search by focusing on the boundary
- For some evolutionary operators the bound violation probability remains large, i.e. the repairing rule plays an important role in the search process
- It might reduce the population diversity

# Bound constraints handling

Projection

## Main idea

- The components which violate the constraints are replaced with the closest bound

$$x_j = \begin{cases} a_j & \text{if } x_j < a_j \\ b_j & \text{if } x_j > b_j \end{cases}$$

- It preserves the previous search direction

## Advantages

- Easy implementation and small costs
- Useful when the optimum is on the bounds

## Disadvantages

- It introduces a bias in the search by focusing on the boundary
- For some evolutionary operators the bound violation probability remains large, i.e. the repairing rule plays an important role in the search process
- It might reduce the population diversity

# Bound constraints handling

Projection

## Main idea

- The components which violate the constraints are replaced with the closest bound

$$x_j = \begin{cases} a_j & \text{if } x_j < a_j \\ b_j & \text{if } x_j > b_j \end{cases}$$

- It preserves the previous search direction

## Advantages

- Easy implementation and small costs
- Useful when the optimum is on the bounds

## Disadvantages

- It introduces a bias in the search by focusing on the boundary
- For some evolutionary operators the bound violation probability remains large, i.e. the repairing rule plays an important role in the search process
- It might reduce the population diversity

# Bound constraints handling

Reflection

## Main idea

- For each component which violates the bounds iterate:

$$x_j = \begin{cases} b_j - (x_j - b_j) & \text{if } x_j > b_j \\ a_j + (a_j - x_j) & \text{if } x_j < a_j \end{cases}$$

until $x_j \in [a_j, b_j]$.

## Advantages

- It might increase the diversity

## Disadvantages

- For some evolutionary operators the bound violation probability remains large, i.e. the repairing rule plays an important role in the search process
- It looses the information on the search direction

# Bound constraints handling

Reflection

## Main idea

- For each component which violates the bounds iterate:

$$x_j = \begin{cases} b_j - (x_j - b_j) & \text{if } x_j > b_j \\ a_j + (a_j - x_j) & \text{if } x_j < a_j \end{cases}$$

until $x_j \in [a_j, b_j]$.

## Advantages

- It might increase the diversity

## Disadvantages

- For some evolutionary operators the bound violation probability remains large, i.e. the repairing rule plays an important role in the search process
- It looses the information on the search direction

# Bound constraints handling

Reflection

## Main idea

- For each component which violates the bounds iterate:

$$x_j = \begin{cases} b_j - (x_j - b_j) & \text{if } x_j > b_j \\ a_j + (a_j - x_j) & \text{if } x_j < a_j \end{cases}$$

  until $x_j \in [a_j, b_j]$.

## Advantages

- It might increase the diversity

## Disadvantages

- For some evolutionary operators the bound violation probability remains large, i.e. the repairing rule plays an important role in the search process
- It looses the information on the search direction

# Summary

- the constraint handling methods can be combined with any metaheuristic approach
  - some of the handling methods (penalty method, multi-objective reformulation) do not require any change in the algorithm
  - other methods (feasibility rules, stochastic ranking, $\epsilon$-constraints) interferes only with the selection step
- the bounding-box constraint handling methods (resampling, reinitialization, projection, reflection) are based on changes in the reproduction step (e.g. new elements are created such that they satisfy the constraints)