Lecture 9: Constraint Handling

CSE5012: Evolutionary Computation and Its Applications

Xin Yao

CSE, SUSTech

24 April 2023

Outline of This Lecture

What Is Constrained Optimisation

The general problem we considered here can be described as:

$$\min_{\mathbf{x}}\{f(\mathbf{x})\}$$

subject to

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

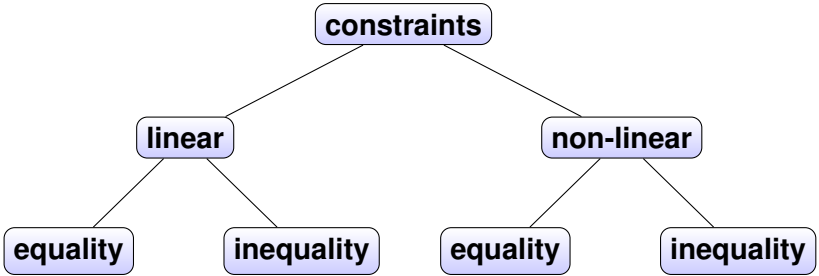
$$h_j(\mathbf{x}) = 0, \quad j = 1, 2, \ldots, p$$

where

- $\mathbf{x}$ **is the** n**-dimensional vector,** $\mathbf{x} = (x_1, x_2, \ldots, x_n)$**;**
- $f(\mathbf{x})$ **is the objective function;**
- $g_i(\mathbf{x})$ **is the inequality constraint;**
- $h_j(\mathbf{x})$ **is the equality constraint.**

Denote the whole search space as $\mathcal{S}$ **and the feasible space as** $\mathcal{F}$**,** $\mathcal{F} \subset \mathcal{S}$**. It is important to note that the global in** $\mathcal{F}$ **might not be the same as that in** $\mathcal{S}$**.**

Different Types of Constraints



- **Linear constraints are relatively easy to deal with.**
- **Nonlinear equality constraints can be hard to handle.**

Constraint Handling Techniques

- **Indirect constraint handling (before EA):**
  - The **penalty function approach** converts a constrained problem into an unconstrained one by introducing a *penalty term* into the objective function.
- **Direct constraint handling (during EA):**
  - The **repair approach** maps (repairs) an infeasible solution into a feasible one.
  - The **purist approach** rejects all infeasible solutions in search.
  - Using **specialised representations and operators** to ensure all candidate solutions are feasible.
  - Using **decoder functions** to replace the original mapping to take the constraints into account.
- The **separatist approach** considers the objective function and constraints separately.
- The **hybrid approach** mixes two or more different constraint handling techniques.

Outline of This Lecture

Penalty Function Approach: Introduction

NewObjectiveFunction = OriginalObjectiveFunction + PenaltyCoefficient * DegreeOfConstraintViolation

The general form of the exterior penalty function method:

$$\psi(\mathbf{x}) = f(\mathbf{x}) + \left(\sum_{i=1}^{m} r_i G_i(\mathbf{x}) + \sum_{j=1}^{p} c_j H_j(\mathbf{x}) \right),$$

where

- $\psi(\mathbf{x})$ **is the new objective function to be minimised,**
- $f(\mathbf{x})$ **is the original objective function,**
- r_i **and** c_j **are penalty factors (coefficients),**
- $G_i(\mathbf{x}) = (\max(0, g_i(\mathbf{x})))^{\beta}$ **and** $H_j(\mathbf{x}) = \max(0, |h_j(\mathbf{x})|^{\gamma})$**.**
- β **and** γ **are usually chosen as 1 or 2.**

Exterior and Interior Penalty Functions [1]

- **Exterior penalty functions:**
  - **penalise the infeasible solutions only.**
- **Interior penalty functions: penalise all solutions based on distance from the constraint boundary in order to encourage exploration of this region.**

Issues & Challenges of Penalty Function Approach

$$\psi(\mathbf{x}) = f(\mathbf{x}) + \left(\sum_{i=1}^{m} r_i G_i(\mathbf{x}) + \sum_{j=1}^{p} c_j H_j(\mathbf{x}) \right)$$

1. **Assumption:** **it is possible to evaluate an infeasible solution.**
   $\rightarrow$ **This is not always the case :(**

2. **Balance** **between exploration of the infeasible region and not wasting computational budget.**
   $\rightarrow$ **Design of suitable penalty function and distance metric.**

Penalty Function Approach: Techniques

Static Penalties **The penalty function is pre-defined and fixed during evolution.**

- **Extinctive penalties.**
- **Binary penalties.**
- **Distance-based penalties. $\leftarrow$ Best [2]**

Dynamic Penalties **The penalty function changes according to a pre-defined sequence, which often depends on the generation number.**

Adaptive and Self-Adaptive Penalties **The penalty function changes adaptively. There is no fixed sequence to follow.**

Static Penalty Functions

$$\psi(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} r_i \left(G_i(\mathbf{x})\right)^2$$

where r_i's are pre-defined and fixed [2].

- **Equality constraints can be converted into inequality ones:**

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

  where $\varepsilon > 0$ is a small number.
- **Simple and easy to implement.**
- **Requires rich domain knowledge to set r_i's.**
- **r_i's can be divided into a number of different levels. When to use which is determined by a set of heuristic rules.**

Dynamic Penalty Functions: An Example

General principle: The large the generation number, the larger the penalty coefficient. *Why?*

Joines and Houck's method [3]:

$$\psi(\mathbf{x}) = f(\mathbf{x}) + (Ct)^{\alpha} \left(\sum_{i=1}^{m} |g_i(\mathbf{x})|^{\beta} + \sum_{j=1}^{p} |h_j(\mathbf{x})|^{\beta} \right),$$

where C, α and β are constants defined by the user, e.g., $C = 0.5, \alpha = 1, 2, \beta = 2$, t is the generation number.

- **Many different forms of dynamic penalties are possible, e.g., $\frac{t}{T}$. The best one can be difficult to find.**
- **Different coefficients can be used for inequality and equality constraints.**

Dynamic Penalties: Generalisation

$$\psi(\mathbf{x}) = f(\mathbf{x}) + r(t)\sum_{i=1}^{m} G_i^2(\mathbf{x}) + c(t)\sum_{j=1}^{p} h_j^2(\mathbf{x}),$$

where $r(t)$ and $c(t)$ are two penalty coefficients.

Polynomials a_i and b_i are user-defined parameters.

$$r(t) = a_0 + a_1 t + a_2 t^2 + \cdots, \quad c(t) = b_0 + b_1 t + b_2 t^2 + \cdots.$$

Exponentials a and b are user-defined parameters.

$$r(t) = e^{at}, \quad c(t) = e^{bt}.$$

Hybrid

$$r(t) = e^{a_0 + a_1 t + a_2 t^2 + \cdots}, \quad c(t) = e^{b_0 + b_1 t + b_2 t^2 + \cdots}.$$

Adaptive Penalties: An Example

Bean and Hadj-Alouane's method:

$$\psi(\mathbf{x}) = f(\mathbf{x}) + \lambda(t) \left(\sum_{i=1}^{m} G_i^2(\mathbf{x}) + \sum_{j=1}^{p} |h_j(\mathbf{x})| \right),$$

where λ is updated at generation t as follows:

$$\lambda(t) = \begin{cases} \frac{1}{\beta_1}\lambda(t), & \textbf{if case 1,} \\ \beta_2\lambda(t), & \textbf{if case 2,} \\ \lambda(t), & \textbf{otherwise,} \end{cases}$$

where case 1 (or 2) indicates that the best individual in the last k generations was always feasible (or infeasible), $\beta_2 > \beta_1 > 1$.

What does the technique mean?

Fitness Function and Selection

- **Let $\Phi(\mathbf{x}) = f(\mathbf{x}) + rG(\mathbf{x})$, where $G(\mathbf{x}) = \sum_{i=1}^{m} G_i(\mathbf{x})$ and $G_i(\mathbf{x}) = \max\{0, g_i(\mathbf{x})\}$.**
- **Given two individuals $\mathbf{x}_1$ and $\mathbf{x}_2$. Their fitness values will now be determined by $\Phi(\mathbf{x})$.**
- **Because fitness values are used primarily in selection,**

$$\textbf{Changing fitness} \iff \textbf{changing selection probabilities}.$$

When r Plays an Important Role

$$\Phi(\mathbf{x}_1) < \Phi(\mathbf{x}_2)$$

means

$$f(\mathbf{x}_1) + rG(\mathbf{x}_1) < f(\mathbf{x}_2) + rG(\mathbf{x}_2).$$

1. $f(\mathbf{x}_1) < f(\mathbf{x}_2)$ **and** $G(\mathbf{x}_1) < G(\mathbf{x}_2)$**:** r **has no impact on the comparison.**
2. $f(\mathbf{x}_1) < f(\mathbf{x}_2)$ **and** $G(\mathbf{x}_1) > G(\mathbf{x}_2)$**: Increasing** r **will eventually change the comparison.**
3. $f(\mathbf{x}_1) > f(\mathbf{x}_2)$ **and** $G(\mathbf{x}_1) < G(\mathbf{x}_2)$**: Decreasing** r **will eventually change the comparison.**

In essence, different r**'s lead to different rankings of individuals in the population.**

Penalty Functions Demystified

Penalty function
$\Longrightarrow$ Fitness transformation
$\Longrightarrow$ Rank change (selection change)

Why not change the rank directly in an EA?

Outline of This Lecture

Stochastic Ranking - Self-adaptive [6, 7]

- **Stochastic ranking is a special rank-based selection scheme that handles constraints.**
- **There is no need to use any penalty functions.**
- **It's self-adaptive since there are few parameters to set.**

1. **DO the following until there is no more change in the ranking.**
2. **FOR** $i := 1$ **TO** $\mu - 1$ **DO**
   - 2.1 $u := U(0, 1)$**, where** u **is a uniformly distributed random number;**
   - 2.2 **IF** $G(\mathbf{x}_i) = G(\mathbf{x}_{i+1}) = 0$ **OR** $u \leq P_f$ **THEN**
     IF $f(\mathbf{x}_i) > f(\mathbf{x}_{i+1})$ **THEN** $swap(\mathbf{x}_i, \mathbf{x}_{i+1})$**;**
   - 2.3 **ELSE**
     IF $G(\mathbf{x}_i) > G(\mathbf{x}_{i+1})$ **THEN** $swap(\mathbf{x}_i, \mathbf{x}_{i+1})$**;**

*P_f **indicates the probability of using the objective function for comparison in ranking.**

The Role of P_f

$P_f > 0.5$: **Most comparisons are based on $f(\mathbf{x})$ only. Infeasible solutions are likely to occur.**

$P_f < 0.5$: **Most comparisons are based on $G(\mathbf{x})$ only. Infeasible solutions are less like to occur, but the solutions might be poor.**

In practice, P_f is often set between 0.45 and 0.5.

Question

Is feasibility guaranteed in this case?

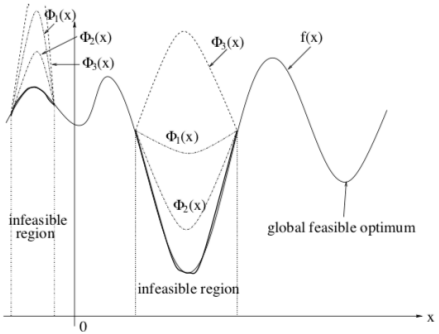
Penalties and Fitness Landscape Transformation

- **Recall:**

  NewObjectiveFunction = OriginalObjectiveFunction +

  PenaltyCoefficient * DegreeOfConstraintViolation

- **Different penalty functions lead to different new objective functions.**



Well Transformed Objective Functions

1. $\Phi_1(\mathbf{x})$ **on the previous page is well transformed because the global optimum of** $\Phi_1(\mathbf{x})$ **is the feasible optimum we want.**

2. $\Phi_2(\mathbf{x})$ **is poorly transformed because its global optimum is infeasible. This is a typical case of *under-penalisation*.**

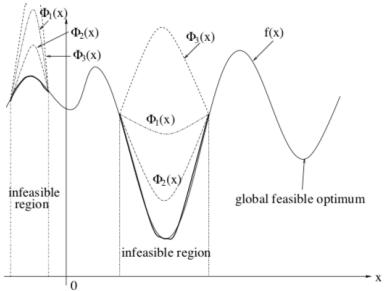
Question

Does it mean the penalty function should be really large?

What Would be an Appropriate Penalty

Question

Is $\Phi_2(\mathbf{x})$ a good choice? Why or why not?

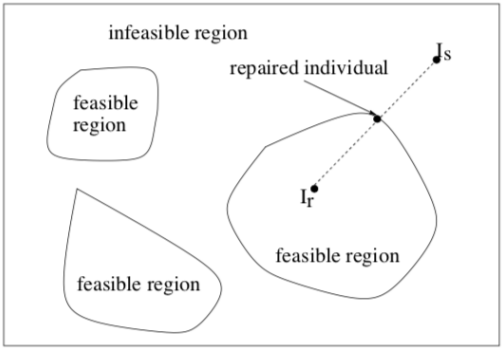


The balance between objective function and penalties is important.

Outline of This Lecture

Repair Approach to Constraint Handling

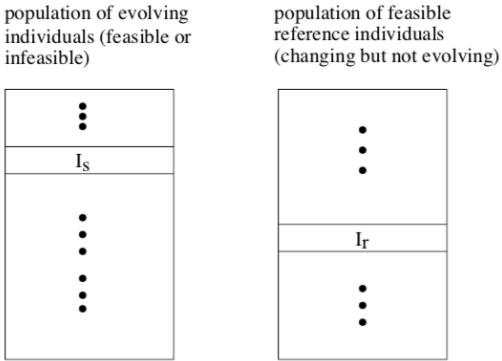
Instead of modifying an EA or fitness function, infeasible individuals can be repaired into feasible ones.
$\Rightarrow$ EA $+$ local search aiming at minimising constraint violation.



Repairing Infeasible Individuals

Let

- I_s **be an infeasible individual (search point) and**
- I_r **a feasible individual (reference point).**



Repairing Algorithm

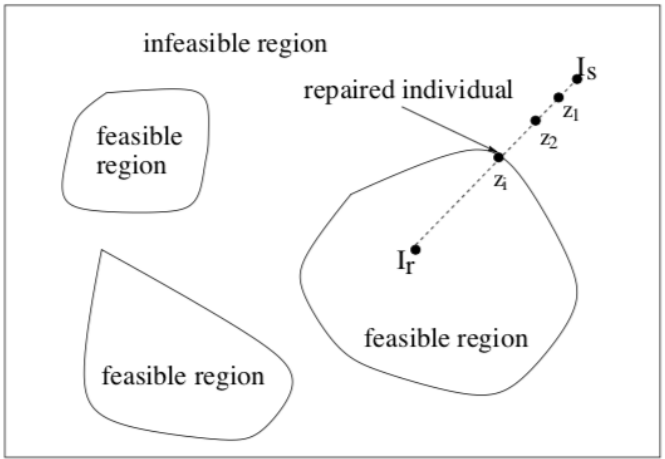
1. **Select a reference individual I_r.**
2. **Create a sequence of candidate individuals z_i between I_s and I_r:**

$$z_i = a_i I_s + (1 - a_i) I_r,$$

where $0 < a_i < 1$ can be generated deterministically or at random. The process of creating z_i stops when z_i is feasible (i.e., when the first feasible z_i is found).

3. **Let this z_i be the repaired individual of I_s. Its objective function value will be used as I_s's fitness value.**
4. **Replace I_s by z_i with probability P_r. (Even if I_s is not replaced by z_i, its fitness is still that of z_i's.)**
5. **If z_i is better than I_r, replace it.**

Repairing Algorithm: Visualisation



Repairing Algorithm: Implementation Issues

How to find initial reference individuals?

1. **Preliminary exploration.**
2. **Human knowledge.**

How to select I_r?

1. **Uniformly at random.**
2. **According to the fitness of I_r.**
3. **According to the distance between I_r and I_s.**

How to determine a_i?

1. **Uniformly at random between 0 and 1.**
2. **Using a fixed sequence, e.g., $\frac{1}{2}, \frac{1}{4}, \ldots$**

How to choose P_r?

A small number, usually < 0.5.

Repairing, Lamarckian Evolution and Baldwin Effect

1. **The repairing approach is closely linked to the wider issues regarding Lamarckian evolution and Baldwin effect. This is due to two characteristics:**
   - 1.1 **Repairing occurs only within one generation.**
   - 1.2 **Repaired individuals do not always replace the original ones.**
2. **Individual repairing $\Longleftrightarrow$ individual learning.**
3. **Although learned characteristics (i.e., repairs done) are not inherited, (individual) learning helps and guide (population) evolution.**
4. **Learning appears to smooth out bumps in the fitness landscape and thus help evolution.**

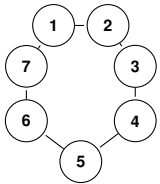
Outline of This Lecture

Specialised Representations and Operators [1, 5]

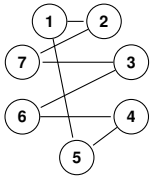
- **Design specialised representations and operators to ensure all candidate solutions are feasible.**
- **Thus the search is reduced and limited to the feasible regions $\mathcal{S}_F$ of the solution space $\mathcal{S}$.**

Example: Travelling Salesman Problem (TSP)

- **Permutation representations for TSPs.**
- **Crossover using edge map. ← Always yields feasible solutions!**



Parent 1: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$



Parent 2: $(x_1, x_2, x_7, x_3, x_6, x_4, x_5)$

	Edge Map
1:	2, 7, 5
2:	1, 3, 7
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

Example: Arithmetic Crossover

- **Given**
  - d **is the individual/chromosome length,**
  - **two parents** $\mathbf{x}_1$ **and** $\mathbf{x}_2$**, and**
  - **a weight** $\alpha \in [0, 1]$**:**

$$Offspring\ 1 : x'_i = \alpha x_{1i} + (1 - \alpha)x_{2i}, \quad i \in \{1, \ldots, d\}$$

$$Offspring\ 2 : x''_i = \alpha x_{2i} + (1 - \alpha)x_{1i}, \quad i \in \{1, \ldots, d\}$$

- **If** $\mathbf{x}_1$ **and** $\mathbf{x}_2$ **feasible solutions, the offspring are feasible solutions in convex search space.**

Issues of Using Specialised Representations and Operators

- **Need to respect 2 assumptions:**
  1. **All of the feasible region is capable of being represented.**
  2. **Any feasible solution can be reached from any other solution by one or more applications of the mutation operator.**
- **Not suitable for all types of constraints.**
- **Assumption 2 is difficult to meet.**
- **Checking whether a solution is feasible or not may be time consuming :(**

Outline of This Lecture

Decoder Functions [1, 4]

A set of mappings from the genotype space $\mathcal{S}'$ to the feasible regions $\mathcal{S}_F$ of the solution space $\mathcal{S}$, with which:

- **every $z \in \mathcal{S}'$ maps to a single solution $s \in \mathcal{S}_F$;**
- **every $s \in \mathcal{S}_F$ has at least one representation $s' \in \mathcal{S}'$;**
- **every $s \in \mathcal{S}_F$ must have the same #representations in $\mathcal{S}'$ (need not be 1).**

Example: Knapsack Problem

- **Usually, a solution is represented as a bit string of size n if n items.**
- **Interpret a 1 as "take this item if possible", otherwise 0.**
- **If the cost limit is reached after considering j of the n genes, then it is irrelevant what values the rest take, and so 2^{n-j} strings all map onto the same solution.**
- **Example:**
  - **Assume 5 items and total cost limit$= 35$.**
  - **The cost limit is reached when items 1 and 2 are taken, or 2 and 3 are taken.**
  - **Solutions s_1 and s_2 map to $s' = 110**$.**
  - **Solutions s_3 and s_4 map to $s'' = 011**$.**

Cost/item	**10**	**24**	**8**	**26**	**16**
s_1	**1**	**1**	**0**	**1**	**1**
s_2	**1**	**1**	**0**	**0**	**0**
s_3	**0**	**1**	**1**	**1**	**1**
s_4	**0**	**1**	**1**	**0**	**1**

Issues of Using Decoder Functions

- **The genotype-phenotype mapping is complex.**
- **The fitness landscape associated with the search space is also highly complex.**
  - **In the previous Knapsack example, the changes at the left-hand end of the bit string have higher impact on fitness changes than those at the right-hand end.**
- **It could be difficult to specify exactly the common features the recombination operators are supposed to be preserving.**
- **Generally introduce a lot of redundancy into the original genotype space when the new mapping is many-to-one.**

Summary

1. **Adding a penalty term to the objective function is equivalent to changing the fitness function, which is in turn equivalent to changing selection probabilities.**
2. **It is easier and more effective to change the selection probabilities directly and explicitly. Stochastic ranking enables us to do this.**
3. **There are other constraint handling techniques than the penalty method.**
4. **We have covered numerical constraints only. We have not dealt with constraints in a combinatorial space.**

Outline of This Lecture

References for This Lecture

[1] **Agoston E Eiben, James E Smith, et al.** ***Introduction to evolutionary computing.*** **Vol. 53. Springer, 2003.**

[2] **David E Goldberg. "Genetic algorithms in search". In:** ***Optimization, and MachineLearning*** **(1989).**

[3] **Jeffrey A Joines and Christopher R Houck. "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's". In:** ***Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence.*** **IEEE. 1994, pp. 579–584.**

[4] **Slawomir Koziel and Zbigniew Michalewicz. "A decoder-based evolutionary algorithm for constrained parameter optimization problems". In:** ***International Conference on Parallel Problem Solving from Nature.*** **Springer. 1998, pp. 231–240.**

[5] **Zbigniew Michalewicz. "A survey of constraint handling techniques in evolutionary computation methods.". In:** ***Evolutionary programming*** **4 (1995), pp. 135–155.**

[6] **Thomas P. Runarsson and Xin Yao. "Stochastic ranking for constrained evolutionary optimization". In:** ***IEEE Transactions on evolutionary computation*** **4.3 (2000), pp. 284–294.**

[7] **Thomas Philip Runarsson and Xin Yao. "Search biases in constrained evolutionary optimization". In:** ***IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*** **35.2 (2005), pp. 233–243.**

Reading List for Next Lecture

1. **U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," IEEE Transactions on Evolutionary Computation, vol. 17, no. 3, pp. 368–386, 2013.**
2. **——, "Reusing genetic programming for ensemble selection in classification of unbalanced data," IEEE Transactions on Evolutionary Computation, vol. 18, no. 6, pp. 893–908, 2014.**
3. **P. Wang, K. Tang, T. Weise, E. Tsang, and X. Yao, "Multiobjective genetic programming for maximizing roc performance," Neurocomputing, vol. 125, pp. 102–118, 2014.**