

Heuristic Algorithms

Master's Degree in Computer Science/Mathematics

Roberto Cordone

DI - Università degli Studi di Milano



Schedule: Thursday 14.30 - 16.30 on MS-Teams

Friday 14.30 - 16.30 on MS-Teams

Office hours: on appointment

E-mail: roberto.cordone@unimi.it

Web page: <https://homes.di.unimi.it/cordone/courses/2020-ae/2020-ae.html>

Ariel site: <https://rcordoneha.ariel.ctu.unimi.it>

Genetic algorithms

Algorithm GeneticAlgorithm($I, X^{(0)}$)

$\Xi^{(0)} := \text{Encode}(X^{(0)}); x^* := \arg \min_{x \in X^{(0)}} f(x); \quad \{ \text{Best solution found so far} \}$

For $g = 1$ *to* n_g *do*

$\Xi := \text{Selection}(\Xi);$

$\Xi := \text{Crossover}(\Xi);$

$x_c := \arg \min_{\xi \in \Xi} f(x(\xi));$

If $f(x_c) < f(x^*)$ *then* $x^* := x_c;$

$X^{(g)} := \text{Mutation}(X^{(g)});$

$x_m := \arg \min_{\xi \in \Xi} f(x(\xi));$

If $f(x_m) < f(x^*)$ *then* $x^* := x_m;$

EndFor;

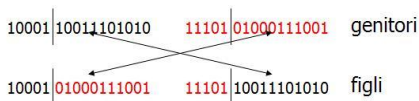
Return $(x^*, f(x^*));$

Crossover

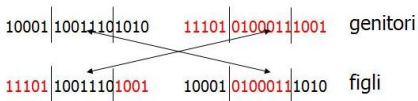
The **crossover** operator combines $k \geq 2$ individuals to generate other k

The most common ones set $k = 2$ and are

- **simple crossover:**
 - extract a random position with uniform probability
 - split the encoding in two parts at the extracted position
 - exchange the final parts of the encodings of the two individuals



- **double crossover:**
 - extract two positions at random with uniform probability
 - split the encoding in three parts at the extracted positions
 - exchange the extreme parts of the encodings of the two individuals



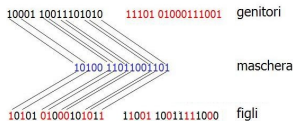
Generalizing, one obtains the

- α points crossover:
 - extract α positions at random with uniform probability
 - split the encoding in $\alpha + 1$ parts at the extracted positions
 - exchange the odd parts of the encodings of the two individuals (first, third, etc. . .)

For small values of α , this implies a **positional bias**:
symbols close in the encoding tend to remain close

To cancel this bias, one can adopt the

- **uniform crossover**:
 - build a random binary vector $m \in U(\mathbb{B}^n)$ (“mask”)
 - if $m_i = 1$ exchange the symbols in position i of the two individuals, if $m_i = 0$ keep them unmodified



Crossover versus Scatter Search and Path Relinking

The crossover operator resembles the recombination phase of *SS* and *PR*

The main differences are that

- ① it recombines the symbols of the encodings, instead of
 - recombining the solutions (*SS*)
 - performing a chain of exchanges on the solutions (*PR*)
- ② it operates on the whole population, instead of only a reference set *R*
- ③ it operates on random pairs of individuals, instead of methodically scanning all pairs of solutions of *R*
- ④ it generates a pair of new individuals, instead of
 - generating a single intermediate solution (*SS*)
 - visiting the intermediate solutions and choosing one (*PR*)
- ⑤ the new individuals enter the population, instead of becoming candidates for the reference set

However, classifying an operator can be a matter of taste

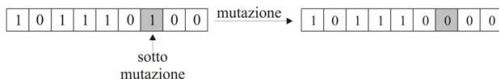
Mutation

The **mutation** operator **modifies an individual to generate a similar one**

- scan encoding ξ one symbol at a time
- decide with probability π_m to modify the current symbol

The kind of modification usually depends on the encoding

- **binary encodings**: flip ξ_i into $\xi'_i := 1 - \xi_i$



- **symbol strings**: replace ξ_c with a random symbol $\xi'_c \in B_c \setminus \{\xi_c\}$ selected with a uniform probability
- **permutations**: there are many proposals
 - **exchange two random elements** in the permutation (*swap*)
 - **reverse the stretch between two random positions** of the permutation
 - ...

Mutation versus exchange heuristics

The mutation operator has strong relations with exchange operations

The main differences are that

- ① it modifies the symbols of an encoding,
instead of exchanging elements of a solution
- ② it operates on random symbols,
instead of exploring a neighbourhood systematically
- ③ it operates on a random number of symbols,
instead of exchanging a fixed number of elements

However, classifying an operator can be a matter of taste

The feasibility problem

If the encoding is not fully invertible, **crossover and mutations sometimes generate encodings that do not correspond to feasible solutions**

We distinguish between

- **feasible encodings** that correspond to feasible solutions
- **unfeasible encodings** that correspond to legal, but unfeasible subsets

The existence of unfeasible encodings implies several disadvantages:

- **inefficiency**: computational time is lost handling meaningless objects
- **ineffectiveness**: the heuristic explores less solutions (possibly, none)
- **design problems**: fitness must be defined also on unfeasible subsets

There are three main approaches to face this problem

- 1 **special encodings and operators** (*avoid or limit infeasibility*)
- 2 **repair procedures** (*turn infeasibility into feasibility*)
- 3 **penalty functions** (*accept infeasibility, but discourage it*)

Special encodings and operators

The idea is to investigate

- **encodings that** (nearly) **always yield feasible solutions**, such as
 - permutation encodings and order-first split-second decoding for partition problems (*CMSTP*, *VRP*, etc. . .)
 - permutation encodings and a constructive heuristic decoding for scheduling problems (*PMSP*, . . .)
- **crossover and mutation operators that maintain feasibility**, such as
 - operators that simulate k -exchanges for the *TSP*

These methods

- tend to closely approximate exchange and recombination heuristics based on the concept of neighbourhood
- give up the idea of abstraction and focus on the specific problem, contrary to classical genetic algorithms

Repair procedures

A **repair procedure** $x_R(\xi)$

- receives an encoding ξ that yields an unfeasible subset $x(\xi) \notin X$
- returns a feasible solution $x_R \in X$

The procedure is applied to each unfeasible encoding $\xi \in \Xi^{(g)}$

- in some methods **the encoding $\xi(x_R(x(\xi)))$ replaces ξ in $X^{(g)}$**
- in other ones **ξ remains in $\Xi^{(g)}$ and $x_R(\xi)$ is used only to update x^***

The first family of methods

- **maintain a population of feasible solutions**

but they introduce

- a strong **bias in favour of feasible encodings**
- a **bias in favour of the feasible solutions most easily obtained** with the repair procedure

Penalty functions: measuring the infeasibility

If the objective function is extended to unfeasible subsets $x \in 2^B \setminus X$, the fitness function $\phi(\xi)$ can be extended to any encoding, but many unfeasible subsets have a fitness larger than the optimal solution

The selection operator tends to favour such unfeasible subsets

To avoid that, the fitness function must combine

- the objective value $f(x(\xi))$
- a measure of infeasibility $\psi(x(\xi))$

$$\begin{cases} \psi(x(\xi)) = 0 & \text{if } x(\xi) \in X \\ \psi(x(\xi)) > 0 & \text{if } x(\xi) \notin X \end{cases}$$

If the constraints of the problem are expressed by equalities or inequalities, $\psi(x)$ can be defined as a weighted sum of their violations

*How to define the weights?
Are they fixed, variable or adaptive?*

Penalty functions: definition of the fitness

The most typical combinations are

- **absolute penalty:** given two encodings ξ and ξ'
 - if both are feasible, the one with smaller f is better
 - if exactly one is feasible, the feasible one is better than the other
 - if both are unfeasible, the one with smaller ψ is better

(suitable for rank and tournament selection)
- **proportional penalty:** the unfeasible encodings have a penalty increasing with the violation

$$\varphi(\xi) = f(x(\xi)) - \alpha\psi(x(\xi)) + M \quad \text{for maximisation problems}$$

$$\varphi(\xi) = -f(x(\xi)) - \alpha\psi(x(\xi)) + M \quad \text{for minimisation problems}$$

where offset M guarantees that $\varphi(\xi) \geq 0$ for all encodings

- **penalty obtained by repair**, that is keep the unfeasible encoding, but derive its fitness from the objective value of the repaired solution

$$\varphi(\xi) = f(x_R(\xi)) \text{ or } \varphi(\xi) = UB - f(x_R(\xi))$$

since usually $f(x_R(x(\xi))) \geq f(x(\xi))$

Proportional penalty functions: weight tuning

Experimentally, it is better to use the smallest effective penalty

- if the penalty is too small, too few feasible solutions are found
- if the penalty is too large, the search is confined within a part of the feasible region

A good value of the parameter α tuning the penalty can be found with

- **dynamic methods:** increase α over time
(*first reach good subsets, then enforce feasibility*)
- **adaptive methods:** update α depending on the situation
 - increase α when unfeasible encodings dominate the population
 - decrease α when feasible encodings dominate
- **evolutionary methods:** encode α in each individual, in order to select and refine both the solution and the algorithm parameter

Memetic algorithms

Memetic algorithms (Moscato, 1989) are inspired by the concept of **meme** (Dawkins, 1989) that is a **basic unit of reproducible cultural information**

- genes are selected only at the phenotypic expression level
- memes also adapt directly, as in Lamarckian evolution

Out of the metaphor, **memetic algorithms combine**

- **“genotypic” operators that manipulate the encodings** (crossover and mutation)
- **“phenotypic” operators that manipulate the solutions** (local search)

In short, **the solutions are improved with exchanges before reencoding**

Several parameters determine how to apply local search

- how often (at every generation, or after a sufficient diversification)
- to which individuals (all, the best ones, the most diversified ones)
- for how long (until a local optimum, beyond, or stopping before)
- with what method (steepest descent, VNS, ILS, etc. . .)

Evolution strategies

They have been proposed by Rechenberg and Schwefel (1971)

The main differences with respect to classical genetic algorithms are:

- the solutions are encoded into **real vectors**
- **a small population of μ individuals generate λ candidate descendants**
(*originally, $\mu = 1$*)
- the new individuals compete to build the new population
 - in the **(μ, λ) strategy** the best μ descendants replace the original population, even if some are dominated
 - in the **$(\mu + \lambda)$ strategy** the best μ individuals overall (predecessors or descendants) survive in the new population
- the mutation operator **sums to the encoding a random noise with a normal distribution of zero average**

$$\xi' := \xi + \delta \text{ with } \delta \in N(0, \sigma)$$

- originally, the crossover operator was not used (*now it is*)