Unit 3. Genetic algorithms

Initialize a random population

Repeat:

- Evaluate population
- Choose parents
- Cross parents
- Mute some descendants
- Select next generation individuals

Until → exact solution
→ approximate solution
→ exceeded nº of iterations

1. Representation: what way we represent a chromosome Ej: binary, matrix,..

   Ej: chromosome 1: 001001100110 $\quad\delta = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$

2. Fitness function: (evaluation): evaluates the quality of a chromosome
   it can be a simple calculation, equation, error ...

3. Population: represents all solutions that exist at any given time. between 20 and 1000 individuals. It never changes during the execution. We initialize it randomly.

4. Parent selection mechanism: it chooses from the population the best adapted to generate next population. An individual is a parent if he has been selected. Better quality individuals are more likely to be chosen.

   - Roulette method: the probability of choosing a chromosome as a parent is $P(C_i) = \frac{g(c_i)}{g(c_1)..g(c_n)}$ so that chromosomes with better fitness values are more likely to be chosen for next generations.

     Only aplicable with positive values

Para invertir restamos cada valor del máximo + una constante

• Ranking method: the probability of a chromosome being chosen, depends on its relative position with respect the others in the population. We order them according to the evaluation and assign probabilities according to:

$$P(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)} \quad \text{where } s \in [1,2] \text{ and } \mu \text{ is } N^{\circ} \text{ of individuals}$$

Chromosomes with the same fitness value occupy the same position in the ranking. It is invariant due to changes in the evaluation function.

Sometimes this method does not induce enough selective pressure so this is used:

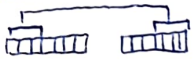
$$P_{exp\text{-}rank}(i) = \frac{1 - e^{-i}}{c} \quad \text{where } c \text{ is chosen so that } sum(P(i)) = 1$$

• Tournament method: it's the simplest one, as we don't need to precalculate all fitness values. We first select k individuals and make them compete according to their fitness function. We repeat the process until completing the required parents. We can do it with replacement (it can be chosen again if chosen once) or without replacement. The probabilities of choosing depend on:

1. Fitness of some chromosome in relation to the other fitnesses.
2. The size k of the tournaments (the larger, the better fitness chromosomes will be chosen)
3. If we use replacement or not (if not, some chromosomes will have 0 chance)

• Stochastic tournament method: were we assign a probability $p \in [0,1]$ to the best k individuals, $p(1-p)$ to the second k best, $p(1-p)^2$ to third ...

If $p=1$ is the deterministic case.

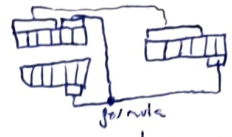
5. Crossing: operator that takes 2 individuals, combines them and returns 1 or 2 descendants. • In binary problems, crossover on a point  (first x components of the first chromosome and last $N-x$ of the other one). You can also take a middle piece and passit to the other which is called crossover over n-points or a uniform crossover, taking some of one and some of the other 
It presents a distributional bias whereas the N-crossing positional biases.

• In integer coding: same as in binary coding

• In floating point coding: if we apply the same methods, we don't introduce new values others than in mutation, so we need to create a new one, by using a parameter $\alpha \in [0,1]$ and calculating the value of the offspring's gene $x' = \alpha x + (1-\alpha) y$. We can use simple crossing where (normally $\alpha = 0.5$)

(x: first parent, y: second parent)

choosing a part, we determine the new values of one side, by applying x' formula and assigning the new values.



We can also use individual arithmetic crossing where we do the crossing just choosing a random position and combined arithmetically. We can also do total arithmetic crossing, crossing the hole chromosomes. In all methods, 2 genes are obtained, depending on the assignation of parent 1 and 2.

- Permutation codings: partially mapped crossover (PMX) looking for adjacency, where we first copy a middle part of first parent by choosing 2 random points, after that we put, the p2 value in the position where the p1 value is, and if it's occupied, we search for the new p1 value position the remaining positions are filled with the P2 elements. Edge crossing: the idea is to preserve as many adjacencies as possible. we create an adjacency table and select an element at random. we remove al references to the current element and examine the list. If there's a common edge, we choose next element, if not, we choose the entry with the shortest list, ties are broken randomly. If we reach an empty list, the other end of the descendant is examined to continue. If not, random choosing It only produces 1 descendant. Order-based crossover: when the order is important. We choose 2 random points of P1 and copy it. Now from that second point, we copy in order the P2 values that have not appeared yet. Cycle crossing: preserves information about the absolute position of the elements. we construct sequences between P1 and P2 so that, we will exchange the cycles with each other to generate 2 descendants. Multiparental crossing: based on frequency, segmentation and recombination, numerical operators but they haven't been proved to upgrade.

6. Mutation: acts on an individual and returns a slightly modified individual (mutant)

- Binary codings: we consider each value independent and change it with a probability between L: lenght chromosome K: population size $\frac{1}{KL} < prob < \frac{1}{L}$
- Integer codings: random reset. the value is changed by another random possible value according to a probability p. Creep mutation: we add or substract (equally possible) a random value selected by a symmetric prob. distribution with respect 0 so that as closer to 0 more probable to be chosen.

- Floating point codings: uniform mutation: value changed by another random value in the interval of the variable, with probability p. (p can change for each position) Non-uniform mutation according to a fixed distribution: we get a Gaussian probability distribution with zero mean and deviation set by user, for each gene, according to a p, we choose a random real value according to the distribution and add it (can be -). If we get out the interval, we set the max / min.
- Permutation codings: exchange mutation: swap two random positions insertion mutation: we choose 2 random and move one towards the other one. Shake mutation: we select a random substring and exchange random positions inverse mutation: we reverse the order of values between 2 random positions.

7. Selection of survivors / substitution: consists of choosing the individuals that will be part of the following population. We do a replacement method to choose the μ individuals between the mutated and crossed ones, and the previous population. Based on age: the oldest individuals are eliminated, Fitness based replacement: taking into account the fitness, we can use any parent selection method and we can combine it with the based on age replacement. Genitor: the worst individuals from the previous population are selected to be replaced, leads to premature convergence but rapid. Elitism: we look for the best fitness individual of the previous population and if it has no better descendants or it is going to disappear, we keep it.

Premature convergence: when individuals of much better fitness than the rest, tend to dominate the population very quickly.
However if they all have very similar fitness, the algorithm will be very slow.

To treat premature we change $f(x)$ by: $f'(x) = \max(f(x) - (\bar{f} - c \cdot \sigma_f), 0)$ Goldberg sigma scaling
To treat slow convergence we can change $f(x)$ by $f(x) = f(x) - b$ (b is min fitness or average of mins in last iterations)

Gray coding: Hamming distance: nº of mismatched positions in binary $d(c_1, c_2) = |p_1 - g_1| + |p_2 - g_2| \ldots$
Gray coding makes binary, that in each step, only 1 value changes. So when using Hamming distance, its distance between 6 and 7 is 1 and also 7 and 8 is 1.

Theoretical foundations of genetic algorithms.

- Schema: hyperplane in the space of solutions. We use symbols such that $11***$ is in $\{0,1\}$ where the 2 positions are fixed and the 3 last can take any other values.

  $o(H)$: order of the schema: number of fixed positions $= 4$

  $\delta(H)$: definitory lenght of the schema: maximal distance between 2 fixed positions $= 4$

- A chromosoma of length l, belongs to 2^l different schemata. 10 belongs to $**$, $1*$, $*0$, 10 (2^2)

  $P_d(H,x)$: probability that an operator x acting over an element of H destroys it

  $P_s(H)$: probability of choosing an element from H.

- $f(H)$: average fitness of chromosomes of H
- $\bar{f}$: average fitness of the chromosomes in the population
- $N(H,t)$: number of individuals of schema H in generation t
- γ: number of individuals chosen for reproduction or mutation

Theorem:

$$N(H,t+1) \geq N(H,t) \cdot \frac{f(H)}{\bar{f}} \left(1 - p_c \frac{\delta(H)}{l-1}\right)(1 - p_m o(H))$$

p_m: prob of mutation
p_c: prob of crossover

Short schemata of low order and with a fitness ($f(H)$) higher than the average ($\bar{f}$), tend to reproduce faster in subsequent generations.

→ If Mean fitness of the chromosomes in the schema is better than the one in all the chromosomes, and the $\delta(H)$ and $o(H)$ are small, the number of individuals of these scheme is going to increase. (or maintain)

It will stop when the number of fixed positions is small ($o(H)$) and the fixed positions are all together (small $\delta(H)$). When this happens, all chromosomes will end up following that schema, and then we will add more information. (fixing a non-fixed value)

Deceiving problem: it's when a high fitness value indicates that you are close to the solution but the solution is far away.

Teórica o dado un population and operator, ¿can we besure it will converge?

Markovian Z matrix:

Represents all possible populations of size n with individuals of length l.

$Z_{r \times N}$ $\quad r \to Z^l$ $\quad N \to$ nº of possible different populations

$z_{y,i}$: number of times chromosome y appears in the i-th population

To assure that a genetic algorithm is convergent one of these need to be true:

- If each individual has a non-null probability of being chosen as a parent
- If each individual has a non-null probability of being chosen as a survivor
- If the mechanism for choosing survivors is elitist (the best ones have more probability of surviving)
- If every solution can be created through the operators with a non-null probability

Global optimum is assured after a finite number of iterations.

Puro teorico, no te va a preguntar como resolverias
- que emplearías solo teoría

Unit 4 Control of parameters

- Memetic algorithm: algorithm based on the evolution of populations which tries to use every available knowledge for doing an heuristic search. Usually this info arises from specific local search algorithms.

- Evolutionary algorithms are good global explorers but bad profiters (good solutions may not be detected). Local search algorithms are good getting good solutions.

- No free lunch theorem: on average all algorithms behave the same way, but if it is very good for a given class of problems, it should behave very bad for another class of problems. Good in a domain → bad in another domain.

- Hibridation: to incorporate more information in the specific problem we are considering. We will use specific heuristics for considering problems as, for instance, local searches. We look for an equilibrium between robustness and reliability (global search) and accuracy (local search).

- Mene → imitation unit, similar to a gene but in cultural evolution

- Memetic algorithms mix: self-improvement periods (local searches), cooperation periods (recombination) and competing periods (selection).

- We have agents instead of chromosones. Selection and replacement are purely competitive. Reproduction (cooperation) generates new agents. The 2 main possible operators are: recombination and mutation.
  - Recombination: involved in cooperation, creates new agents combining the infor. of already existing ones. (Intelligent combination of information)
  - Mutation: allows the inclusion of external info. It partially modifies agents.

Agent → crossing → mutation → local optimization → new agent

There's not a specific way to design memetic algorithms. We can design heuristics but do not guarantee good results.

When is the local search applied?

It may be in the initialization phase, in each generation or after a given nº, after each reproduction or after applying recombination operators,.. but MA must only be used inside the evolutive process.

Over which agents is it applied?

In the whole population, a subset, agents resulting from reproduction...

What do we do with the optimized agent?

2 main models:

Lamarckian: the agent obtained in optimization replaces the optimized agent

Baldwinian: the resulting agent gives its fitness to the optimized agent but the latter does not change.

How do we apply it?

We must get an equilibrium between the frequency of application of the optimizer (breadth) and the intensity of the optimizer (depth).

There are not general rules of which local search algorithm do we use

Unit 5 Gravitational search algorithm

An heuristic algorithm for maximizing or minimizing an objective function (fitness function). Possible solutions are seen as particles in the space. They have a mass which depends on how good they are as solutions (using fitness f.) The particles evolve only because of their gravitational attraction until find solution. We only need to know which function we want to optimize. Randomness is essential. In an iterative way, it tries to find an optimus (max/min) of a function $fit: \mathbb{R}^n \to \mathbb{R}$

We start with N random vectors in $\mathbb{R}^n$ (particles). We also provide for each a vector $\vec{V}_i(t) \in \mathbb{R}^n$ which represents the velocity of the particle. They are fixed randomly (normally near 0). For each iteration we have N particles with their velocity.

1. We assign masses i) evaluating with the fitness function and keep the best and worst values (for maximizing best is the biggest and the other way around)

(minimizing) $$best(t) = \min_{j \in 1..N} fit(\vec{x}_j(t))$$

$$worst(t) = \max_{j \in 1..N} fit(\vec{x}_j(t))$$

ii) we assign a relative mass $m_i(t) = \frac{fit(\vec{x}_i(t)) - worst(t)}{best(t) - worst(t)}$. It's $[0,1]$ where the best particle is given a 1 and the worst one a 0.

iii) we assign a mass M_i by just normalizing the relative mass $m_i(t)$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}$$ (the worst still has a 0)

Fecha:

2. Calculation of the gravitational points forces

i) We define $\vec{F}_{ij}(t)$ as the force of particle j acting over particle i $(i \neq j)$

$$\vec{F}_{ij}(t) = G(t) \frac{M_i(t) \cdot M_j(t)}{R_{ij}(t) + \varepsilon} (\vec{X}_j(t) - \vec{X}_i(t))$$ where R_{ij} is the euclidean (L^2) distance between $\vec{X}_j(t)$ and $\vec{X}_i(t)$ and $\varepsilon > 0$ is a small constant

$R_{ij} = d_2(\vec{X}_i, \vec{X}_j) = \sqrt{(x_i^1 - x_j^1)^2 + (x_i^2 - x_j^2)^2} \dots$ $G(t)$ is a positive constant which can vary on each iteration. Usually required $\lim_{t \to \infty} G(t) = 0$. (As if takes a point $=0$ the algorithm would stop)

ii) Now we calculate the total force over a particle $\vec{X}_i(t)$. We build a vector $(w_1 \dots w_N) \in [0,1]^N$ where w_j is a uniformly distributed random number in the interval $[0,1]$. Now, $\vec{F}_i(t) = \sum_{j=1, i \neq j}^{n} w_j F_{ij}(t)$

→ o sin priorizar nada, es un random simple

This random fact can help bad particles to have a better value.

3. Calculation of the acceleration

The acceleration $\vec{a}_i(t) = \frac{\vec{F}_i(t)}{M_i(t)}$ but if $M_i(t) = 0 \rightarrow \vec{a}_i(t) = G(t) \sum_{i \neq j} \frac{M_j(t)}{R_{ij}(t) + \varepsilon} (\vec{X}_j(t) - \vec{X}_i(t))$

4. Calculation of the new velocities

$V_i(t+1) = p_i \vec{V}_i(t) + \vec{a}_i(t)$ where $p_i \in [0,1]$ is a random number (uniform distribution)

5. Calculation of the new positions

$\vec{X}_i(t+1) = x_i(t) + \vec{V}_i(t+1)$

- Now we iterate 1-5 until:
  - reached max-iterations
  - finding optimum (or $< \varepsilon$ error (prefixed))
  - variation from iteration in position < tolerance (pre-fixed)

It always converge to a steady state. It's usually slow. Solution is given by the particle with best fitness in his steady state.

Converges to velocities = 0 for example

Nombre y Apellidos: ..... DNI: .....
Izen-deiturak: NANa:

Titulación: ..... Asignatura: .....
Titulazioa: Irakasgaia:

Convocatoria: ..... Grupo: .....
Deialdia: Taldea:

$\begin{pmatrix} 0.9 & 0.7 & 0.7 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$ find $\bar{a} = (a_1, a_2, a_3)$ $\bar{b} = (b_1, b_2, b_3)$ such that $\bar{a} \circ \bar{b} = A$

$\bar{X} = (\underbrace{x_1, x_2, x_3}_{\bar{a}}, \underbrace{x_4, x_5, x_6}_{\bar{b}})$

$\bar{X} \to \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \circ (x_4, x_5, x_6) = \begin{pmatrix} \min(x_1, x_4) & \min(x_1, x_5) & \cdots \\ \min(x_2, x_4) & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}$

$fit(\bar{X}) = (\min(x_1, x_4) - 0.9)^2 + \cdots$

1) Initialize 4 particles and 4 velocities

$\bar{X}_1 = (0.2, 0.3 \ldots 0.6)$ $\bar{V}_1 = (0.4 \ldots 0.1)$

$\bar{X}_4 = (0.9, 0.2, \ldots 0.1)$ $\bar{V}_4 = (0.7 \ldots 0.5)$

(mismo fitness en todos → error → cambiar $fit(\bar{x})$)

Assign masses

2) Fitness. $fit(\bar{x}_1) = (\min(\bar{x}_{11}, \bar{x}_{14}) - 0.9)^2 + \ldots = 1.65$ ... $fit(\bar{x}_4) = 1.24$

So $best(0) = 1.24$ and $worst(0) = 2.33$ as we are minimizing error

3) $m_1(0) = \frac{fit(\bar{x}_1) - worst(0)}{best(0) - worst(0)} = \frac{1.65 - 2.33}{1.24 - 2.33} = 0.62$ $m_2(0) \ldots m_4(0) = 1$

(validness)

4) Normalize masses $M_1(0) = \frac{m_1(0)}{\sum_{j=1}^{4} m_j(0)} = \frac{0.62}{0.62 + 0.7 + 1.0 + 1} = 0.27$ ... $M_4(0) = 0.43$

Gravitational forces

5) Calculate forces $\vec{F}_{1,2}(0) = G(0) \cdot \frac{M_1(0) \cdot M_2(0)}{R_{12}(0) + \varepsilon} (\bar{X}_2(0) - \bar{X}_1(0))$

G: constant variable en cada it
R_{12}: euclidean distance
ε: small constant

$R_{12} = \sqrt{(0.2 - 0.2)^2 + (0.4 - 0.3)^2} \ldots = 1.11$ $\vec{F}_{ij} = -\vec{F}_{ji}$

$\bar{F}_{12}(0) = 1 \cdot \frac{0.27 \cdot 0}{1.11 + 0.01} (0, 0.1, -0.1 \ldots) = \bar{0}$ ... $\bar{F}_{14} = 0.61$

We take $w_1 = w_2 = w_3 = w_4 = 1$

Total force $\Rightarrow \bar{F}_1 = w_1 \cdot \bar{F}_{12} + w_2 \cdot \bar{F}_{13} + w_3 \cdot \bar{F}_{13} = (0, -0.01 \ldots 0.6)$ ... $\bar{F}_4 = (\ldots)$

Accelerations

6) Accelerations

If $M_1 \neq 0 \to \bar{a}_1(0) = \frac{\vec{F}_1(0)}{M_1(0)} = (0, -0.04, \cdots)$

If $M_2 = 0 \to \bar{a}_2(0) = \sum_{i \neq 2} \frac{M_j(0)}{R_{2j}(0) + \varepsilon} (\bar{X}_j(0) - \bar{X}_2(0)) = \frac{0.27}{1.12} (0.2 \ldots) + \frac{0.3}{1.01} (\ldots) + \frac{0.43}{1.06} (\ldots)$

$= (0.19, -0.57 \ldots)$

7) New velocities We take $p_1 = \frac{1}{2}$ $p_2 = \frac{1}{3}$ $p_3 = p_4 = 1$

$\vec{V}_1(1) = p_1 \cdot \vec{V}_1(0) + \vec{a}_1(0) = \frac{1}{2} \cdot (0.1\ldots) + (0, -0.04\ldots) = (0.05\ldots)$ ... $\vec{V}_4(1) = \ldots$

8) New positions

$\vec{X}_1(1) = \vec{X}_1(0) + \vec{V}_1(1) = (0.25\ldots)$ ... $\vec{X}_4(1)$

Repeat 2-8 until: reaching the limit of iterations

it holds $\sqrt{(V_i^1(t))^2 + \ldots + (V_i^n(t))^2} < tol$ (defined previously)

when calculating new fitnesses $fit(\vec{X}_4(1)) = 0.44$ have upgraded notably

Puede pedir ejercicio pero poco probable

Unit 6. Particle swarm optimization algorithms

- PSO algorithms imitate the behaviour of particles in the nature. From a set of possible solutions, we move them through the space till we get the optimal one.
- They are usually easier than genetic as they don't have crossing or mutation operators. They also start from random solutions and actualize population by generations.
- Each solution is a particle. We have a fitness function to optimize and for each particle a velocity. We store 2 data, the best fitness value it has found (pbest) and a gbest value with the best found fitness value.

1. Initialization: we choose the number of particles N, we fix 3 parameters (they may be functions): w, k_p, k_g. We initialize randomly a position $X_i(0)$ and its velocity $V_i(0)$ for each particle. Moreover $pbest_i = X_i(0)$. We initialize gbest with the position of $X_i(0)$ which minimizes fitness.

2. Development: to go from generation t to $t+1$ we do for each particle:
   - update its velocity:
     $$V_i(t+1) = w \cdot V_i(t) + k_p (X_i(t) - pbest_i) + k_g (X_i(t) - gbest)$$
   - update its position:
     $$X_i(t+1) = X_i(t) + V_i(t+1)$$
   - If $fit(X_i(t+1)) < fit(pbest_i) \rightarrow pbest_i = X_i(t+1)$
     If $fit(pbest_i) < fit(gbest) \rightarrow gbest = pbest_i$

   Stop conditions: nº iterations / finding optimum / stabilization of solutions

3. Solution = gbest

- PSO have strong tendency towards local optima. To avoid it, instead of the global optimum gbest, use the optimum of subswarm (the m particles closest to the considered particle in each iteration.)
- Normally 10-40 particles (10, in complex 100-200). Normally $k_p = k_g = 2$ but $\in [0, 4]$. If velocity increases too much, use maximal velocity V_m so when happens, $V_i = V_m$.

$V_{maximal}$