EVOLUTIONARY ALGORITHMS

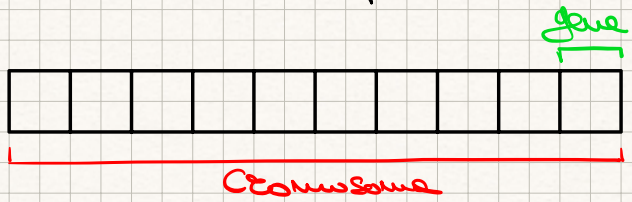
Sono algoritmi che cercano di riprodurre il modo in cui gli organismi evolvono in natura. Di base si cerca di imitare come, presi uno o più cromosomi genitori, si può produrre un cromosoma figlio.
Questo per cercare di riprodurre l'evoluzione di un organismo che si adatta al luogo in cui vive. In natura questa evoluzione è descritta in tre fasi:

1) Selezione degli individui che produrranno prole
2) Ricombinazione degli individui per produrre prole
3) Mutazione casuale del materiale genetico.

In pratica, partendo da diverse soluzioni iniziali, si cerca di combinarle per raggiungere l'ottimo. Vengono inoltre aggiunti elementi di disordine random.

CROMOSOMA

È una soluzione candidata, formata da GENI.



Gli algoritmi genetici presentano le stesse fasi del corrispettivo naturale:

1) Selezione
   Dati un insieme di cromosomi random, scegliere i migliori per la soluzione del problema

2) Ricombinazione

Ricombiniamo fra loro i cromosomi migliori scelti in fase di selezione per cercare figli che fanno un passo verso l'ottimo.

3) Mutazione

Serve cambiare casualmente qualche gene, poiché partendo da cromosomi casuali, probabilmente non sono completi e quindi esiste ancora materiale genetico inesplorato. Evitiamo così ottimi locali.

Pseudo-Codice

Pseudo-codice di un algoritmo genetico:

```
t = 0
P(t) = InizializzaPopolazioneCromosomi ().
ValutaFitness ( P(t) )
while ( Stopping-Condition )
{
    MP = SelezionaCromosomi (P(t))
    Popp = Ricombina (MP)
    Popp = Muta ( Popp )
    P(t+1) = SelezionaDa (Popp, P(t))
    t++
}
```

```
function [x_opt, f_opt, best_fit] = GA_BE_1D(f,a,b,generations,chrsize,popsize,pmutallgenes,displayAndPlot)
    % Generate the initial population
    Popolation = BE_initpop(chrsize,popsize);

    % Computing the fitness of each individual in the initial population
    fit_Pop = BE_evaluatefitness(f, Popolation, a, b);

    % Initialise the vector with the best fitness four at each generation
    best_fit = zeros(1,generations);
    best_fit(1) = min(fit_Pop);

    for t = 2:generations
        % 1. Create the Mating Pool (MP) using binary tournment selection (the MP is of the the same size as P)
        MP = binarytournement(Popolation, fit_Pop, popsize);
        % 2. Reshuffle the Mating Pool
        MP = MP(randperm(popsize),:);
        % 3. Apply the crossover operator to each pair in the MP, to obtain the offspring population Q
        Q = false(popsize,chrsize);
        for i=1:2:popsize
        Q([i,i+1],:) = BE_XOVER_singlepoint(MP(i,:),MP(i+1,:));
        end
        % 4. Apply mutation operator to obtain the mutated version of Q
        for i=1:popsize
        Q(i,:) = BE_MUT_unifall(Q(i,:),pmutallgenes);
        end
        % 5. Evaluate the fitness of Q
        fit_Q = BE_evaluatefitness(f,Q,a,b);
        % 6. Compute R as the union of P and Q
        R = [Popolation; Q];
        fit_R = [fit_Pop; fit_Q];
        % 7. Compute the new population P, by taking the best individuals from R
        [~,ind] = sort(fit_R,"ascend");
        new_P = R(ind(1:popsize),:);
        fit_new_P = fit_R(ind(1:popsize));
        % 8. Let the new population new_P become the current population and do the same for the fitness
        P = new_P;
        fit_Pop = fit_new_P;
        % 9. Update best_fit
        [best_fit(t), ind] = min(fit_Pop);
    end

    [f_opt, ind] = min(fit_Pop);
    x_opt = logical2real(P(ind,:),a,b);
end
```

STOPPING CONDITION

1) Indicare un massimo numero di generazioni
2) Valore di fitness da raggiungere. Ovvero quanto ci vogliamo avvicinare all'ottimo.
3) Indicare il numero di generazioni consecutive dove non ci sono cambiamenti nella popolazione.
4) Terminare se il valor medio di fitness supera una certa threshold rispetto all'individuo migliore.

SELEZIONE

Non è scontato che da due genitori buoni nasca un figlio migliore. Né che da due non buoni nasca un figlio peggiore

SELEZIONE A ROULETTE

A ogni individuo viene assegnata una probabilità di selezione proporzionale alla sua fitness.

SELEZIONE A TORNEO

La più usata è la K-tournament selection, dove si prendono K individui casuali e il migliore viene selezionato.

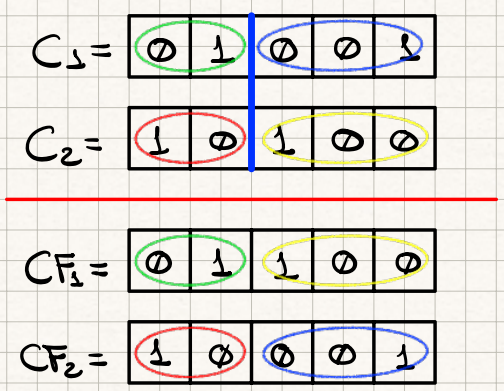
RICOMBINAZIONE

CROSS-OVER PER LA CODIFICA BINARIA

Tratto da uno o più cromosomi genitori e genera figli.

Single-point crossover

Il più semplice crossover. Presi due genitori, li taglio in due parti e intercambio le componenti di ogni genitore per creare due figli



Cococcioni > MATLAB > LAB_02 > BE_XOVER_singlepoint.m

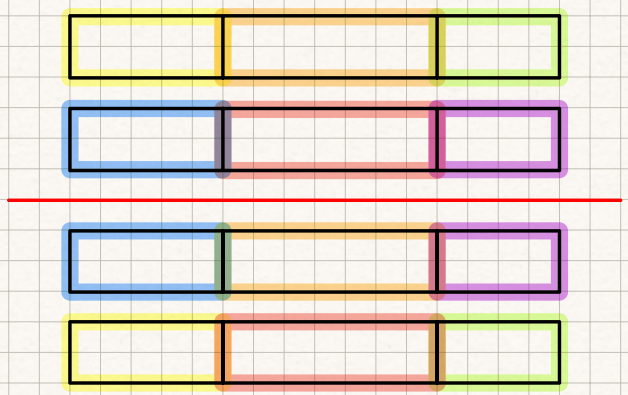
```
function [offsprings, splitpoint] = BE_XOVER_singlepoint(parent1, parent2)
% This function computes the single point crossover
    len = length(parent1);
    splitpoint = randperm(len-2,1)+1;
    offsprings = false(2,len);

    offsprings(1,1:splitpoint) = parent1(1:splitpoint);
    offsprings(1,splitpoint+1:end) = parent2(splitpoint+1:end);

    offsprings(2,1:splitpoint) = parent2(1:splitpoint);
    offsprings(2,splitpoint+1:end) = parent1(splitpoint+1:end);
end
```

Two-Points Crossover

Qui taglio i genitori in due punti, generando tre parti di ogni genitore da combinare



Uniform Crossover

Consiste nello scambiare singoli geni dei due genitori in modo casuale

MUTAZIONE

Si sceglie casualmente un gene del nuovo set di cromosomi e lo si flippa con una certa probabilità.

```
i = random(0, GenLEN)
if (P(i) > threshold)
    C[i] = ~C[i]
```

CODIFICA REALE

Finora abbiamo trattato i geni come binari. Vediamo cosa succede se sono reali:

$$[x_1, x_2, \ldots, x_m] : \forall x_i \in \mathbb{R}$$

RICOMBINAZIONE

1) Crossover discreto

È l'analogo del crossover uniforme del caso binario. Dove abbiamo una probabilità di ricombinare il gene i-esimo del genitore 1 con l'i-esimo del genitore 2. Se la probabilità supera una certa soglia allora possiamo provare a ricombinare.

$C_1 =$	3.5	...	1.7
$C_2 =$	-7.9	...	4.5
$P =$	0.7	...	0.3
$O_1 =$	-7.9	...	1.7
$O_2 =$	3.5	...	4.5

2) Arithmetic Crossover

$$O_1[i] = \alpha P_1[i] + (1-\alpha) P_2[i]$$
$$O_2[i] = \alpha P_2[i] + (1-\alpha) P_1[i]$$

Con $\alpha \in [0, 1]$ random. È una media pesata con la somma dei pesi $\alpha + (1-\alpha) = 1$.

Qui α cambia per ogni gene.

3) Convex Crossover

$$O_1[i] = \alpha P_1[i] + (1-\alpha) P_2[i]$$
$$O_2[i] = \alpha P_2[i] + (1-\alpha) P_1[i]$$

Uguale a quello aritmetico solo che α è fissato e uguale per ogni gene.

MUTAZIONE

Per mutare ogni gene possiamo usare la distribuzione gaussiana:

$$C = [g_1, \dots, g_n]$$

Ogni gene sarà mutato come segue:

$$M_i = g_i + \sigma_i \underbrace{\mathcal{N}(\phi, 1)}_{\text{Gaussiana}} \rightarrow \text{mutazione non-uniforme}$$

Se invece usiamo una mutazione uniforme:

$$M_i = g_i + \Delta_i \underbrace{\mathcal{U}(-1, 1)}_{\text{Distribuzione uniforme}}$$