UNIVERSITAT DE BARCELONA




## Algorísmica

# Algorismes de cerca

**Mireia Ribera, Jordi Vitrià**

## Algorismes de cerca

El concepte de **cerca** inclou diferents conceptes:

- Cerca d’un determinat element en una llista (màx, x=“a”, el que compleix una certa condició, etc.).
- Cerca d’un determinat element en una llista ordenada.
- Cerca en un arbre.
- Cerca en un graf.
- Satisfacció de restriccions.
- Etc.

Ens centrarem en la cerca en llistes.

## Algorismes de cerca: cerca lineal

L'algorisme que implementa amb una estratègia de força bruta la cerca d'un element en una llista es diu **cerca seqüencial o lineal.**

```
def linsearch(list,ele):
     i=0
     while i<len(list) and list[i] != ele:
          i += 1
     if i<len(list): return i
     else: return -1
```

La complexitat de l'algorisme és $O(n)$ en el pitjor cas!

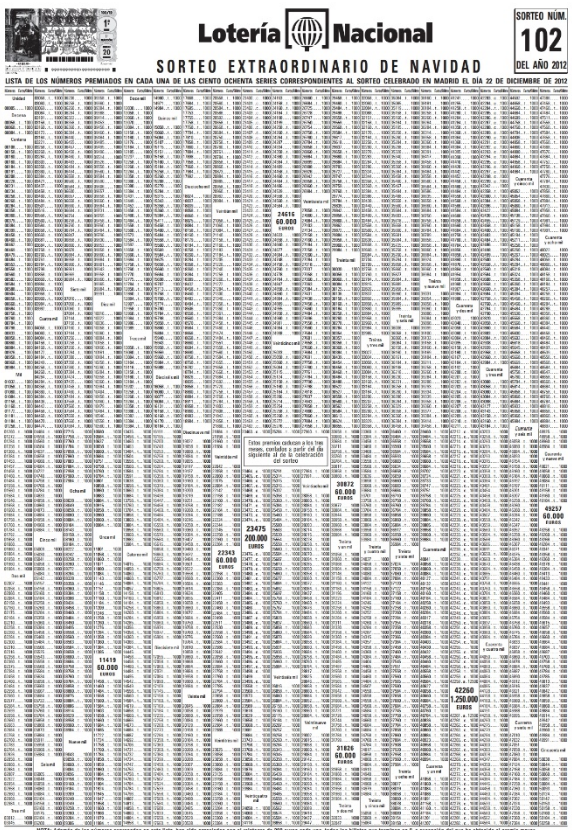
## Algorismes de cerca: cerca lineal

Podem fer una petita millora si afegim l'element que busquem al final de la llista:

```
def linsearch(list,ele):
    list.append(ele)
    i=0
    while list[i] != ele:
        i += 1
    if i<len(list)-1: return i
    else: return -1
```

# Algorismes de cerca: cerca binaria

I si la llista està ordenada (un diccionari, els nombres de la loteria, etc.) ho podem fer millor?


Lotería Nacional

SORTEO NÚM. 102 DEL AÑO 2012

SORTEO EXTRAORDINARIO DE NAVIDAD

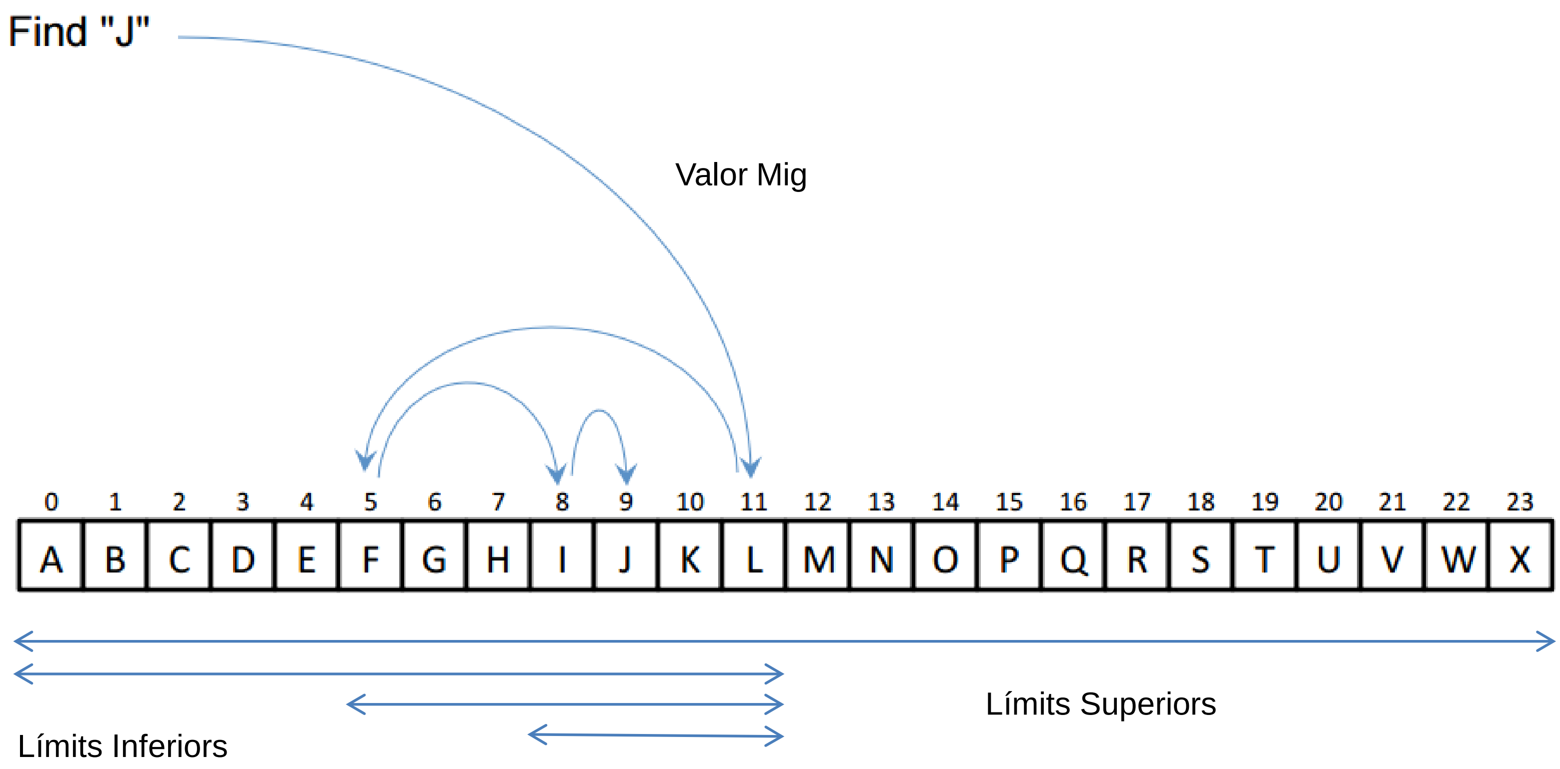
LISTA DE LOS NÚMEROS PREMIADOS EN CADA UNA DE LAS CIENTO OCHENTA SERIES CORRESPONDIENTES AL SORTEO CELEBRADO EN MADRID EL DÍA 22 DE DICIEMBRE DE 2012


**plumber** /ˈplʌmə(r)/ *n* [C] person whose job is to fit and repair water pipes
**plumbing** /ˈplʌmɪŋ/ *n* [U] **1** system of water pipes, tanks, etc in a building **2** work of a plumber
**plume** /pluːm/ *n* [C] **1** cloud of sth that rises into the air **2** large feather
**plummet** /ˈplʌmɪt/ *v* [I] fall suddenly and quickly from a high level: *House prices have ~ed.*
**plump** /plʌmp/ *adj* having a soft, round body; slightly fat ● **plump** *v* [T] ~ **(up)** make sth larger, softer and rounder: ~ *up the pillows* [PV] **plump for sb/sth** (*infml*) choose sb/sth ▶ **plumpness** *n* [U]
**plunder** /ˈplʌndə(r)/ *v* [I,T] steal things from a place, esp during a war ● **plunder** *n* [U] **1** act of plundering **2** things that have been stolen, esp during a war
**plunge** /plʌndʒ/ *v* [I,T] (cause sb/sth to) move suddenly forwards and/or downwards: *The car ~d into the river.* ◇ *He ~d his hands into his pockets.* ● **plunge** *n* [C, usu sing] sudden movement downwards or away from sth; decrease [IDM] **take the plunge** (*infml*) finally decide to do sth important or difficult ▶ **plunger** *n* [C] part of a piece of equipment that can be pushed down
**pluperfect** /ˌpluːˈpɜːfɪkt/ *n* (*gram*) = THE PAST PERFECT (PAST[1])
**plural** /ˈplʊərəl/ *n* [usu sing] *adj*
**p.m.** /ˌpiː ˈem/ *abbr* after 12 o'clock noon
**pneumatic** /njuːˈmætɪk/ *adj* **1** filled with air: *a ~ tyre* **2** worked by air under pressure: *a ~ drill* ▶ **pneumatically** /-kli/ *adv*
**pneumonia** /njuːˈməʊniə/ *n* [U] serious illness affecting the lungs
**PO** /ˌpiː ˈəʊ/ *abbr* **1** = POST OFFICE (POST[1]) **2** = POSTAL ORDER (POSTAL) ■ **P'O box** (*also* **'post office box**) *n* [C] used as a kind of address, so that mail can be sent to a post office where it is kept until it is collected
**poach** /pəʊtʃ/ *v* **1** [T] cook fish or an egg without its shell in water that is boiling gently **2** [I,T] illegally hunt animals, birds or fish on sb else's property **3** [T] take from sb/sth dishonestly; steal sth ▶ **poacher** *n* [C] person who illegally hunts animals, birds or fish on sb else's property
**pocket** /ˈpɒkɪt/ *n* [C] **1** small bag sewn into a piece of clothing so that you can carry things in it **2** small bag or container fastened to sth so that you can put things in it, eg in a car door or handbag **3** [usu sing] amount of money that you have to spend: *He had no intention of paying out of his own ~.* **4** small separate group or area [IDM] **in/out of pocket** (*esp GB*) having gained/lost money as a result of sth ● **pocket** *v* [T] **1** put sth into your pocket **2** keep or take

## Algorismes de cerca: cerca binaria

La **cerca binaria** ho fa comparant l'element cercat *K* a l'element central de la llista: si hi ha correspondència ja l'hem trobat, sinó, busquem a la subllista que correspon.

$$\underbrace{A[0] \ldots A[s-1]}_{\text{Buscarem aquí si } K<A[s]} \quad \overset{K}{\overset{\updownarrow}{A[s]}} \quad \underbrace{A[s+1] \ldots A[n-1]}_{\text{Buscarem aquí si } K>A[s]}$$




Com acaba l'algorisme si J no hi és?

# Algorismes de cerca: cerca binaria

```
def recbinsearch(k, nums, low, high):

# low, high defineixen els limits de la llista
# on buscar, així no cal crear noves llistes

      if low>high: return -1
      mid = (low + high) // 2
      items = nums[mid]
      if items == k: return mid
      elif k < items:
            return recbinsearch(k,nums,low,mid-1)
      else: return recbinsearch(k,nums,mid+1,high)

recbinsearch(3,[1,2,3,4,5,6,7,8,9], 0, 8)

>>> 2
```

# Algorismes de cerca: cerca binaria

Anem a veure com funcionaria per $K$=70.

| Índex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Valor | **3** | **14** | **27** | **31** | **39** | **42** | **55** | **70** | **74** | **81** | **85** | **93** | **98** |
| It 1 | *l* | | | | | | *m* | | | | | | *h* |
| It 2 | | | | | | | | *l* | | *m* | | | *h* |
| It 3 | | | | | | | | *l,m* | *h* | | | | |

## Algorismes de cerca: cerca binaria

Per analitzar la seva complexitat calcularem el nombre de vegades que la **clau de la cerca, *K***, es compara amb un element de la llista.

En el **pitjor dels casos** (quan l'element no hi és), tenim aquesta relació de recurrència:

$$C_{pitjor} = 1\left(\frac{n}{2}\right) + 1$$

Segons el teorema Màster això és $O(\log_2 n)$: per una llista de 1.000.000 elements són 20 comparacions!

## Algorismes de cerca: cerca binaria

Evidentment és un algorisme **recursiu**, però és pot implementar fàcilment de forma **no recursiva**.

```
def binsearch(nums, K):
    low = 0
    high = len(nums)-1
    while low <= high:
        mid = (low + high) // 2
        if nums[mid] > K: high = mid - 1
        elif nums[mid] < K: low = mid + 1
        else: return mid
    return -1
```

## Algorismes de cerca: cerca binaria

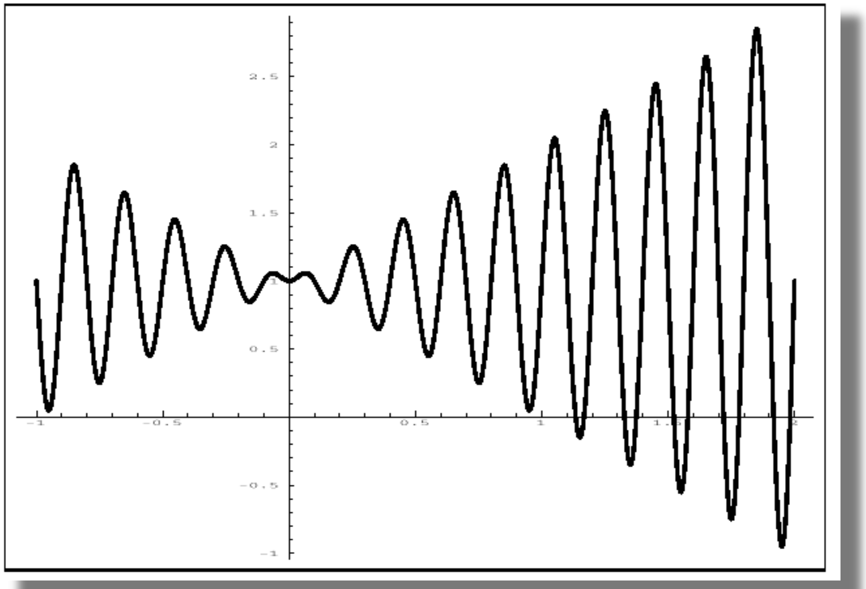
El **cas promig** és més difícil d’analitzar, però es pot demostrar que és només una mica millor que el pitjor cas (tot i que del mateix ordre).

**Observació**: Si tenim una llista desordenada de mida n i només hem de buscar un element (o pocs), apliquem una cerca exhaustiva. Però si hem de fer moltes cerques (de l’ordre de n), val la pena ordenar-la primer i fer cerca binària dels elements després!

# Altres algorismes de cerca (funcions)

Imaginem ara que tenim un **vector no ordenat**, com pot ser el corresponent als valors discrets d'una **funció multimodal**.

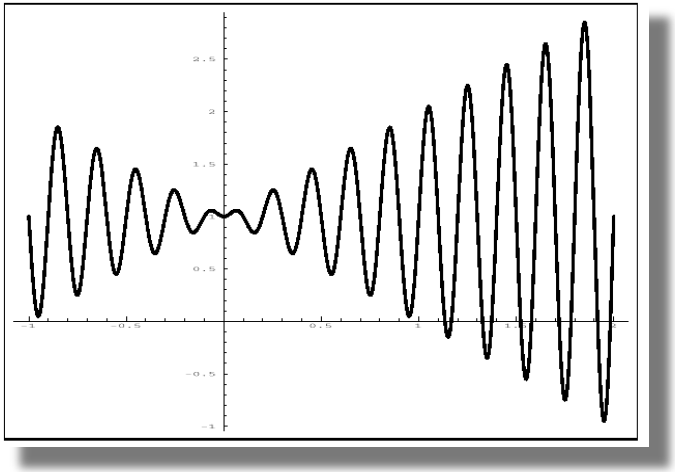
Com busquem el màxim? Quin tipus de cerca hi podem aplicar?



## Algorismes de cerca

Podem aplicar-hi **cerca exhaustiva**:

```
def func1d(x): #funció multimodal de la que buscarem max
    import math
    y = x * math.sin(10*math.pi*(x))+1.0
    return y
```



## Algorismes de cerca

La funció `range` només genera enters

```
def frange(start, stop, step):
    current = start
    while current < stop:
        yield current
        current += step
```

```
def linsearchfunc1d():
    x=0.0      #punt x que dóna valor max
    maxim=0.0  #func1d(x)
    for i in frange(-1.0,2.0,0.01):
        if func1d(i)>maxim:
            maxim=func1d(i)
            x=i
    print(maxim)
```

En aquest cas analitzem 300 punts

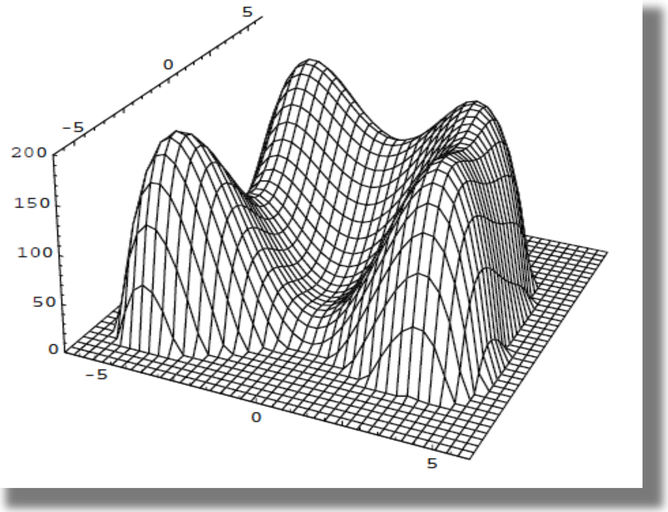
La complexitat és O(n) on n depèn de la precisió que volem

# Algorismes de cerca: cerca aleatòria

Si el nombre de punts a mostrejar és molt gran tenim un problema!

En aquest cas, amb precisió 0.01 analitzaríem 300x300 punts

Si necessitéssim una precisió 0.000001 analitzaríem 3000000x3000000 punts

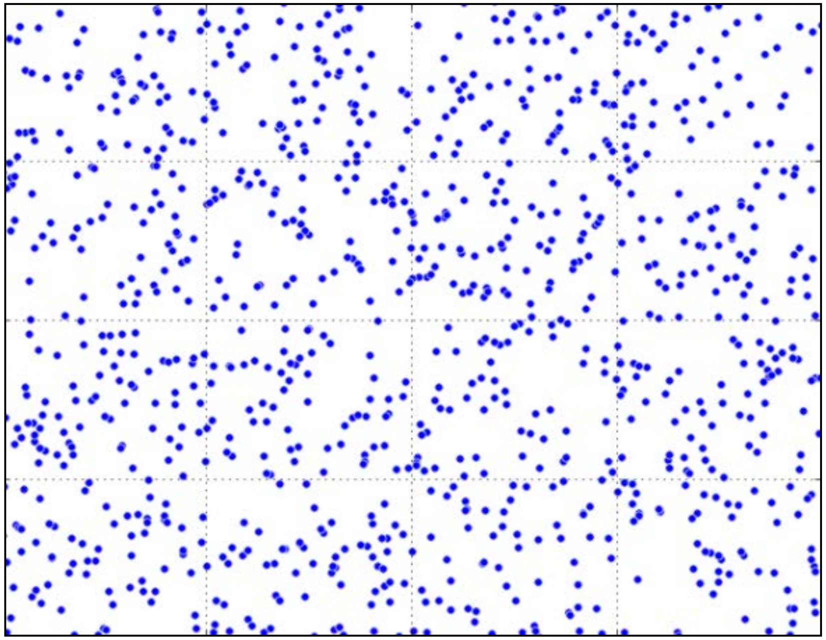



No podem fer cerca exhaustiva!

## Algorismes de cerca: cerca aleatòria

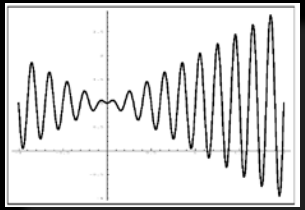
Té sentit fer una **cerca aleatòria**? (= anar generant nombres de forma aleatòria dins del rang de les variables i quedar-se el màxim).

Mostra aleatòria sobre el pla.



# Algorismes de cerca: cerca aleatòria

```
def rsearchfunc1d():
    import random
    x=0.0
    maxim=0.0
    for i in range(1000):
        xtemp = (random.random()*3.0)-1.0
        if func1d(xtemp)>maxim:
            maxim=func1d(xtemp)
            x=xtemp
    return maxim
```



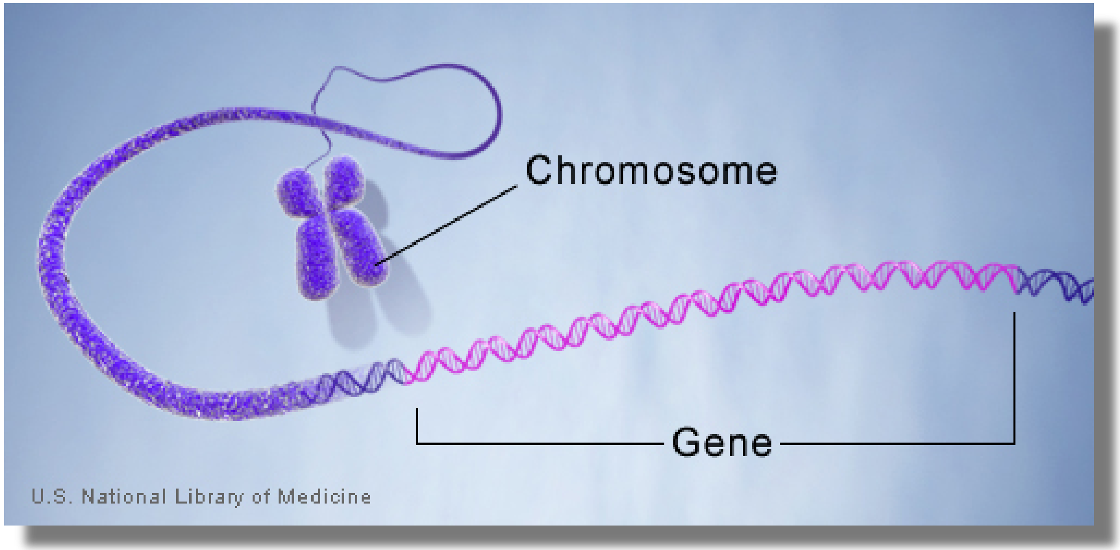
| 100 proves | 1.000 proves | 100.000 proves |
|---|---|---|
| 2.77824636954 | **2.85027049997** | 2.8502736913 |
| 2.76633333502 | 2.84068071726 | 2.85026970833 |
| 2.49830837751 | 2.82585079483 | 2.85026429332 |
| 2.84180575738 | 2.82441879719 | 2.8502737353 |
| 2.67472858999 | 2.84078409458 | 2.8502710006 |
| **2.84721009237** | 2.83748038425 | 2.85026302851 |
| 2.60189299072 | 2.84883426411 | **2.85027375351** |
| 2.81619415008 | 2.8497277592 | 2.85023546116 |
| 2.81493367995 | 2.84184730168 | 2.85027214716 |
| 2.64975396079 | 2.84990510016 | 2.8502538051 |

## Algorismes de cerca: cerca aleatòria

En general, la cerca totalment aleatòria **no és una bona solució**: tenim el cost de la cerca afitat, però depèn molt de l'aleatorietat i té un **resultat molt semblant, si no equivalent, a la cerca lineal, per força bruta**.

# Algorismes de cerca: cerca amb algorismes genètics.

Anem a veure un tipus **d'algorisme aproximat** que ens fa una cerca, amb **un cert component aleatori** més intel·ligent, de l'espai de solucions: la cerca basada en **algorismes genètics**.



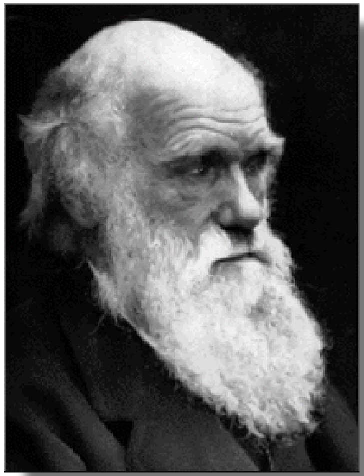

# Algorismes genètics

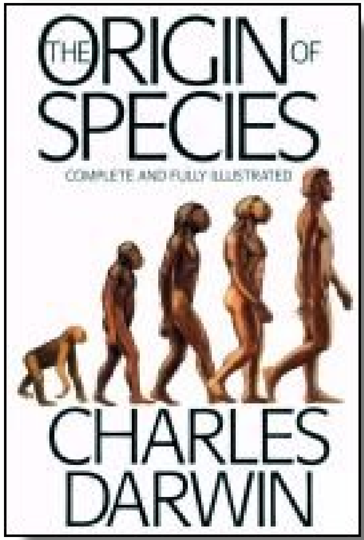
## Algorismes de cerca: cerca amb algorismes genètics.

### Precedents

El terme **algorismes genètics** s'utilitza per a referir-se a una família bastant àmplia de models computacionals de càlcul basats en els mecanismes d'evolució biològica.

La idea de **selecció natural** va ser introduïda per Charles Darwin el 1859 dins del seu llibre "L'Origen de Les Espècies".






Aquesta idea pot servir d'analogia per a construir mètodes de **cerca** en problemes d'optimització combinatòria i mètodes d'**aprenentatge**.

## Algorismes de cerca: cerca amb algorismes genètics.

# Precedents

Darwin va assentar les bases del principi d'evolució per selecció natural amb les següents idees:

- Cada individu tendeix a passar els seus trets característics a la seva descendència.

- Tot i així, la natura produeix individus amb trets diferents.

- Els individus més adaptats tendeixen a tenir més descendència, i a la llarga, la població tendeix a ser "millor".

- Sobre grans períodes, l'acumulació de canvis pot produir espècies totalment noves, adaptades al seu entorn.

A més a més la natura disposa d'una sèrie de mecanismes reguladors externs a aquest procés però igualment interessants: el mecanisme de diversitat, els paràsits, les organitzacions socials, etc.

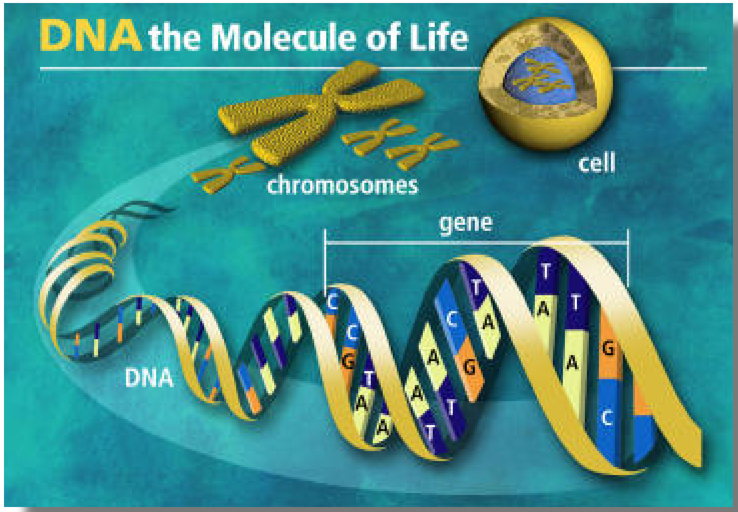
# Algorismes de cerca: cerca amb algorismes genètics.

## Precedents

Els mecanismes biològics que fan possible l'evolució són avui coneguts. A la natura, podem veure com la transmissió de la informació genètica (genoma) es fa a través de la reproducció sexual. Aquest procediment permet als descendents ser diferents dels seus antecessors, tot i que conservant la majoria de trets. El mecanisme sobre el que està basat això es troba a nivell molecular, i consisteix en l'aparellament de cromosomes (lloc on trobem el genoma), l'intercanvi d'informació, i la posterior partició. D'això n'hi direm **creuament**. La probabilitat de que dos individus es creuin depèn de la seva **adaptació** al medi.

Per inspiració d'aquests mecanismes usarem terminologia de biologia per als nostre problemes:

- Gens
- Genoma
- Cromosomes
- Creuaments i mutacions.
- Funció d'adaptació.
- Mecanismes correctors/moduladors: diversitat, parasitisme, organització social, etc.



## Algorismes de cerca: cerca amb algorismes genètics.

El **cicle normal** d'un algorisme genètic és:

- **avaluar** l'adaptació de tots els individus de la població (amb la **funció d'adaptació**). Aquesta funció incorpora l'objectiu del problema.
- **crear** una nova població mitjançant reproducció fent servir:
  - **creuament** +
  - **mutació** dels cromosomes dels individus + descartar la població antiga (hem d'assegurar que el resultat d'aplicar els operadors genera possibles solucions al problema.)
- **iterar** sobre la nova població.

Cada una de les iteracions d'aquest cicle es coneix com **generació**.

## Disseny d’algorismes genètics

Caldrà decidir algunes qüestions:

1. Quina és la funció d'adaptació?
2. Com representarem els individus/solucions?
3. Com seleccionarem els individus per reproduir-se?
4. Com creuarem i mutarem els individus?
5. Quina és la probabilitat de mutació?
6. Necessitem mecanismes moduladors (p.e. diversitat)?

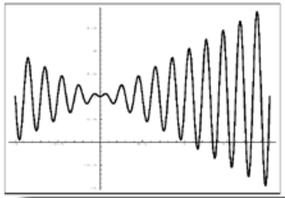
Algorismes de cerca: **cerca amb algorismes genètics.**

## Funció d'adaptació

La funció d'adaptació és **pròpia de cada problema** que volem resoldre.

En el problema que hem posat com a exemple, la funció d'adaptació és el valor de f(x):

```
(math.sin(10*math.pi*(x))+1.0)
```



Per tant, la màxima adaptació correspon al màxim d'aquesta funció multimodal

El problema del viatjant de comerç és un problema candidat a ser resolt amb algorismes genètics.

La funció d’adaptació seria 1/d, on d és la distància recorreguda (i així un valor de la funció alt és una bona solució).

Un cromosoma representaria un circuit que és potencialment solució del problema.

**Algorismes de cerca: cerca amb algorismes genètics.**

## Representació

Normalment es considera que la millor representació dels cromosomes possible és la **binària.**

El creuament, la mutació, i d'altres operacions que es poden utilitzar, són aleshores simples **operacions a nivell de bit**.

## Algorismes de cerca: cerca amb algorismes genètics.

Suposem doncs que tenim una població inicial de quatre individus amb les següents característiques:

| Individu | Valor Adaptació | Probabilitat de Selecció |
|---|---|---|
| 0 0 0 1 1 0 0 1 0 1 1 1 | 8 | 32% |
| 1 1 1 0 1 0 1 0 1 1 0 0 | 6 | 24% |
| 0 0 1 1 1 0 1 0 1 0 0 1 | 6 | 24% |
| 1 1 1 0 1 1 0 1 1 1 0 0 | 5 | 20% |

I suposem que definim la **probabilitat de selecció** com:

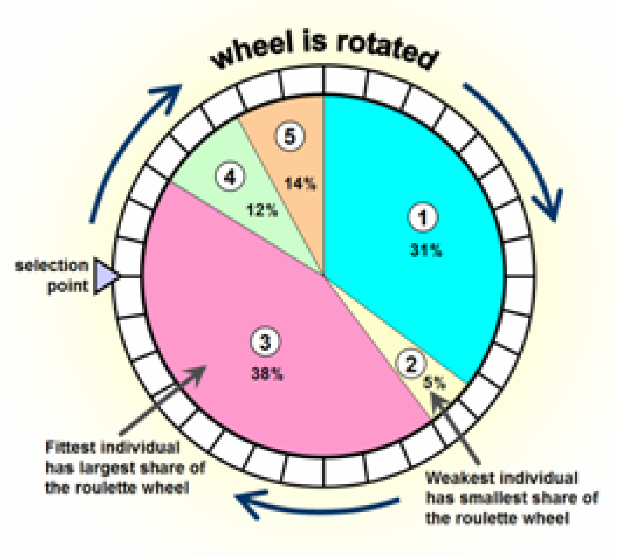
$$f_i = \frac{q_i}{\sum_j q_j}$$

Com els **seleccionem** i els **creuem**?

**Algorismes de cerca: cerca amb algorismes genètics.**

Una primera alternativa per la **selecció** és triar parelles aleatòriament **tenint en compte la seva probabilitat de selecció.**

És com si fem rodar una ruleta i ens va donant individus; l'individu amb més probabilitat de selecció ocupa més posicions a la ruleta.



## **Algorismes de cerca: cerca amb algorismes genètics.**

Imaginem que la selecció ens ha donat aquestes dues parelles:

0 0 0 1 1 0 0 1 0 1 1 1

1 1 1 0 1 0 1 0 1 1 0 0

0 0 1 1 1 0 1 0 1 0 0 1

1 1 1 0 1 1 0 1 1 1 0 0

Població

**1 1 1 0 1 0 1 0 1 1 0 0**

**0 0 0 1 1 0 0 1 0 1 1 1**

**1 1 1 0 1 0 1 0 1 1 0 0**

**0 0 1 1 1 0 1 0 1 0 0 1**

Parelles

I ara, com les **creuem**?

## Algorismes de cerca: cerca amb algorismes genètics.

La forma més simple de creuament és generar un punt de tall aleatòriament i intercanviar:

1 1 1 0 | 1 0 1 0 1 1 0 0
0 0 0 1 | 1 0 0 1 0 1 1 1

Punt de tall seleccionat per la 1a parella.

1 1 1 0 1 0 0 1 0 1 1 1
0 0 0 1 1 0 1 0 1 1 0 0

Aquesta és la nova generació!

1 1 1 0 1 0 1 0 | 1 1 0 0
0 0 1 1 1 0 1 0 | 1 0 0 1

Punt de tall seleccionat per la 2a parella.

1 1 1 0 1 0 1 0 1 0 0 1
0 0 1 1 1 0 1 0 1 1 0 0

## Algorismes de cerca: cerca amb algorismes genètics.

Per a **mutar-los** canviarem el valor d'un quants bits de la població de forma aleatòria.

La probabilitat de que un bit canviï de valor és β i la que probabilitat de no canviar és (1- β), però **sempre β << (1- β)**

**1 1 1 0 1 0 0 1 0 1 1 1**

**0 0 0 1 1 0 1 0 1 1 0 0**

**1 1 1 1 1 0 1 0 1 0 0 1**

**0 0 1 1 1 0 1 0 1 1 0 1**

## Algorismes de cerca: cerca amb algorismes genètics.

### Resum fins ara...

La **funció d'adaptació** depèn del problema que volem resoldre.

La **representació** òptima, és en la majoria de casos i si no hi ha motius fonamentats per dubtar-ho, la **binària**.

La **representació** ha facilitar que el resultat d'aplicar els operadors genètics sigui vàlid.

Les restriccions sobre la representació no formen part de l'algorisme genètic!

L'operació de **creuament** crea dos nous individus seleccionant punts de creuament en els cromosomes seleccionats i intercanviant les seves parts.

L'operació de **mutació** consisteix en la selecció aleatòria d'algun dels gens del cromosoma i el canvi del seu valor. La probabilitat de mutació ha de ser petita (si no ho convertim en cerca aleatòria!).

Algorismes de cerca: **cerca amb algorismes genètics.**

# Probabilitat de supervivència

- El **valor d'adaptació** de cada individu depèn del problema concret.
- La **probabilitat de supervivència a la següent generació** és una **ponderació del valor d'adaptació**, i es pot fer de diverses maneres: el **mètode estàndard** (que ja hem vist), el **mètode d'ordenació**, el **mètode de la diversitat**, etc.
- A partir d'ara suposarem que la generació següent es forma **a partir d'una selecció entre els elements del conjunt format pels cromosomes progenitors i pels cromosomes descendents**, però seguint una **estratègia elitista**: el(s) millor(s) cromosomes passen automàticament (així asseguren que una bona solució no es perd mai).

## Algorismes de cerca: cerca amb algorismes genètics.

## Ponderació del valor d'adaptació.

**Mètode Estàndard**. Donat un cromosoma *i*, aquest és avaluat com a possible solució al problema en qüestió. Com a resultat obté un **valor d'adaptació** $q_i$. Llavors definim la seva **probabilitat de selecció** com:

| Individu | Valor Adaptació | Probabilitat de Selecció |
|---|---|---|
| 0 0 0 1 1 0 0 1 0 1 1 1 | 8 | 32% |
| 1 1 1 0 1 0 1 0 1 1 0 0 | 6 | 24% |
| 0 0 1 1 1 0 1 0 1 0 0 1 | 6 | 24% |
| 1 1 1 0 1 1 0 1 1 1 0 0 | 5 | 20% |

$$f_i = \frac{q_i}{\sum_j q_j}$$

Algorismes de cerca: cerca amb algorismes genètics.

## Ponderació del valor d’adaptació.

## Mètode Estàndard, problemes.

Un dels inconvenients associat al mètode estàndard és el **poc pes que dóna al cromosomes "*dolents*",** fet que els impedeix de passar a les futures generacions, i per tant, transmetre les poques coses que tinguin bones.

Un altre possible inconvenient és que moltes vegades la **funció d’avaluació és qualitativa**: ordena de forma correcta però els seus valors no són precisos.

**Algorismes de cerca: cerca amb algorismes genètics.**

## Ponderació del valor d’adaptació.

## Mètode d'Ordenació.

Per insensibilitzar el mètode de selecció respecte al valor d'adaptació del problema, podem **ordenar els cromosomes segons aquest valor, i els ponderem segons aquesta regla**:

- *Establim un valor aleatori p entre 0 i 1*
- *Al primer cromosoma li assignem aquesta probabilitat*
- *Al segon li assignem la probabilitat p*(1-p)*
- *La probabilitat de l'i-èssim cromosoma és p * (1 – la probabilitat que s’hagi triat algun cromosoma anterior).*

## Algorismes de cerca: cerca amb algorismes genètics.

### Ponderació del valor d'adaptació.

### Mètode d'Ordenació.

0.333 = 1 – (0.667) és la probabilitat de que no hagi sortit el primer cromosoma.

Per exemple, suposem que ***p=0.667***. *Llavors*:

| *Crom (x,y)* | $q_i$ | *Ordre* | *Prob M. Estànd.* | *Prob M. Ordenació* |
|---|---|---|---|---|
| 0001,0100 | 44 | 1 | 0.22 | 0.667 |
| 0011,0001 | 32 | 2 | 0.16 | 0.222= 0.667x0.333 |
| 0001,0010 | 22,5 | 3 | 0.125 | 0.073= 0.667x0.111 |
| 0001,0001 | 1,5 | 4 | 0.075 | 0.025= ... |
| 0111,0101 | 0 | 5 | 0.0 | 0.012 |

Imaginem que aquests cromosomes representen punts del pla i estem buscant el màxim

0.111 = 1 – (0.667 + 0.222) és la probabilitat de que no hagi sortit ni el primer ni el segon cromosoma.

**Algorismes de cerca: cerca amb algorismes genètics.**

## Ponderació del valor d'adaptació.

## Mètode de Diversitat.

Aquest mètode es basa en l'anomenat **principi de diversitat**: <u>és quasi tan bo ser diferent com estar adaptat</u>.

Definim la diversitat d'un grup de cromosomes com:

La distància que usem pot ser des del nombre de bits diferents entre cada cromosoma a una funció definida per l'usuari a partir del coneixement del problema.

$$Div = \sum_i \frac{1}{d_i^2}$$

En el nostre exemple considerarem la distància euclidiana sobre punts del pla. En el cas més general podríem fer servir la distància de Hamming.

on $d_i$ és una **mesura de distància entre cromosomes**.

# Algorismes de cerca: cerca amb algorismes genètics.

La distància que usem pot ser des del nombre de bits diferents entre cada cromosoma a una funció definida per l'usuari a partir del coneixement del problema.

$$Div = \sum_i \frac{1}{d_i^2}$$

En el nostre exemple considerarem la distància euclidiana sobre punts del pla. En el cas general podríem fer servir la distància de Hamming.

La **distància de Hamming** entre dues cadenes de la mateixa longitud és el nombre de posicions diferents. Si considerem cadenes de bits, correspon al nombre de bits que s'han de canviar d'una cadena perquè passi a tenir el valor d'una altra cadena.

## Algorismes de cerca: cerca amb algorismes genètics.

# Ponderació del valor d'adaptació.

## Mètode de Diversitat.

Com l'apliquem?

1. El millor cromosoma passa automàticament a la següent generació (estratègia elitista).
2. Calculem la diversitat de tots els cromosomes respecte als que han passat a la següent generació.
3. Ordenem els cromosomes segons la seva funció d'avaluació.
4. Sumem els nombres que representen l'ordre obtingut per cada cromosoma als passos 2 i 3. I reordenem segons aquest valor.
5. Triem el cromosoma que passa a la següent generació segons el mètode d'ordenació i si queden cromosomes per triar, i tornem al punt 2.

## Algorismes de cerca: cerca amb algorismes genètics.

# Ponderació del valor d'adaptació.

# Mètode de Diversitat.

Exemple:

| *Cromosomes* | $q_i$ |
|---|---|
| 0100 0001 | 100 |
| 0001 0100 | 44 |
| 0011 0001 | 32 |
| 0001 0010 | 22,5 |
| 0001 0001 | 1 |
| 0111 0101 | 0 |

El cromosoma millor passa a la següent generació. En el nostre cas és el cromosoma (0100 0001).

Per tant resten per triar 2 cromosomes entre (0001 0100), (0011 0001),(0001 0010), (0001 0001), (0111 0101).

# Algorismes de cerca: cerca amb algorismes genètics.

## Ponderació del valor d'adaptació.

## Mètode de Diversitat.

Exemple:

Construirem la taula segons el mètode de diversitat segons la diversitat de cada cromosoma respecte al que ja ha passat:

| *Cromosomes* | *diversitat* | *ord div* | *ord est* | *div+est* | *Probab.* |
|---|---|---|---|---|---|
| 0001 0100 | 0.040 | 1 | 1 | 2 | 0.667 |
| 0011 0001 | 0.250 | 5 | 2 | 7 | 0.073 |
| 0001 0010 | 0.059 | 3 | 3 | 6 | 0.222 |
| 0001 0001 | 0.062 | 4 | 4 | 8 | 0.012 |
| 0111 0101 | 0.050 | 2 | 5 | 7 | 0.025 |

En aquest cas, la diversitat és només la inversa de la distància euclidiana 2D de cada cromosoma al que ja ha passat.

La distància que usem pot ser des del nombre de bits diferents entre cada cromosoma a una funció definida per l'usuari a partir del coneixement del problema.

**Algorismes de cerca: cerca amb algorismes genètics.**

# Ponderació del valor d'adaptació.

## Mètode de Diversitat.

**Exemple**:

Llavors triem aleatòriament el següent que passa, i resulta que és el cromosoma (0001 0100). A partir d'aquest moment repetim el procés anterior, però calculant la diversitat respecte al dos cromosomes que ja han passat:

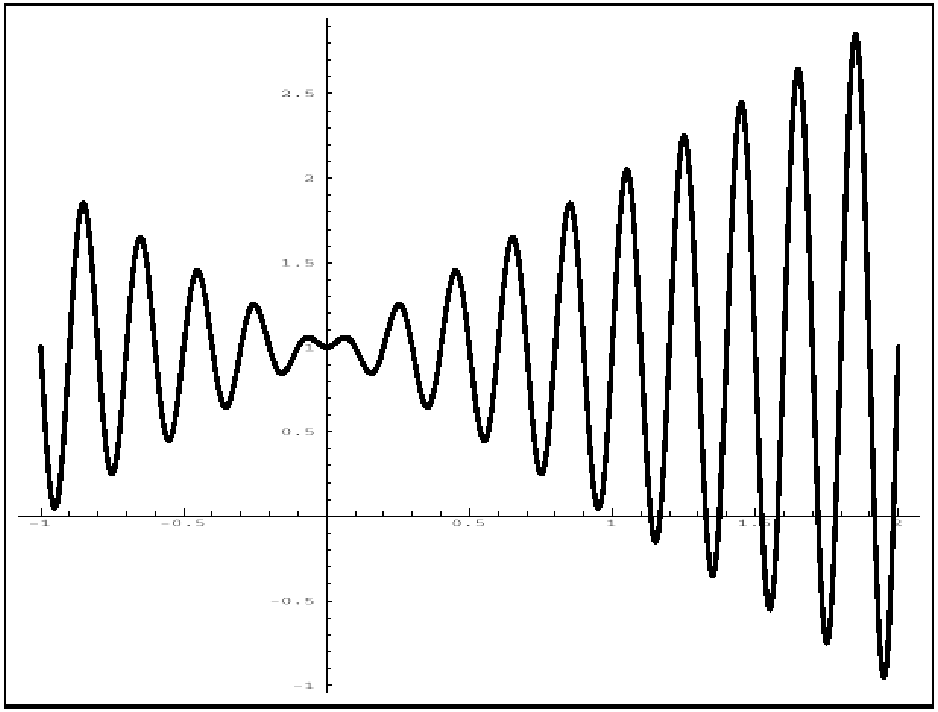
| *Cromosomes* | *diversitat* | *ord div* | *ord est* | *div+est* | *Prob* |
|---|---|---|---|---|---|
| 00110001 | 0.327 | 4 | 1 | 5 | 0.667 |
| 00010010 | 0.309 | 3 | 2 | 5 | 0.222 |
| 00010001 | 0.173 | 2 | 3 | 5 | 0.073 |
| 01110101 | 0.077 | 1 | 4 | 5 | 0.025 |

En el cas d'empats per a l'ordenació resultat de la suma div+ord desempatem segons el valor d'ordenació pura.

## Algorismes de cerca: cerca amb algorismes genètics.

Exemple: Optimització d'una funció multimodal

$$f(x) = x sin(10\pi x) + 1.0$$




El problema és trobar la *x* dins del rang [-1 .. 2] que maximitza *f*.

## Algorismes de cerca: cerca amb algorismes genètics.

Exemple: Optimització d'una funció multimodal

- Utilitzarem un vector binari com a cromosoma per a representar el valor real de la variable *x*. La longitud del vector dependrà del domini i la precisió.
- En el cas estudiat, el domini de la variable x és [-1,2], té longitud 3.
- Suposem que volem 6 decimals (1.000.000 de valors per cada unitat).
- Per tant, necessitem mostrejar el rang en 3.000.000 posicions, o sigui, 22 bits:
  - $2.097.152 = \mathbf{2^{21}} < 3.000.000 < \mathbf{2^{22}} = 4.194.304$

**Algorismes de cerca: cerca amb algorismes genètics.**

Exemple: Optimització d'una funció multimodal

La transformació d'una seqüència binària [$b_{21}$,...,$b_0$] a un nombre real *x* es fa en dos passos:

1. Primer convertim la seqüència de base 2 a base 10:

$$([b_{21},...,b_0])_2 = (\sum_{i=0}^{21} b_i 2^i)_{10} = x'$$

2. Després trobem el nombre real corresponent:

$$x = -1.0 + x' \frac{3}{2^{22}-1}$$

on -1.0 és el límit esquerra de l'interval i 3 la longitud.

**Algorismes de cerca: cerca amb algorismes genètics.**

Exemple: Optimització d'una funció multimodal

Escollim com a població inicial 50 individus de forma aleatòria.

```
# Creem la població inicial
def initpop(n,long):
    import random

    # Generem una poblacio de n cromosomes de longitud long.
    pop = [[0] * long for x in range(n)]
    for i in range(n):
        for j in range(long):
            if random.random()>0.5: pop[i][j] += 1
    return pop
```

**Algorismes de cerca: cerca amb algorismes genètics.**

Exemple: Optimització d'una funció multimodal

La funció d'avaluació serà equivalent a la funció del gràfic *f*:

v1 = (1000101110110101000111), cost(v1) = 1.5886345
v2 = (0000001110000000010000), cost(v2) = 0.078878
v3 = (1110000000111111000101), cost(v3) = 2.250650

```
# Definim la funció d'avaluació d'un cromosoma de len(r) bits.
def cost(r):
    import math
    # Transformem els bits en un valor real a l'interval [-1,2]
    sum=0.0
    for i in range(len(r)): #convertim base 10
        sum = sum + r[i]*(2**i)
    x = -1.0 + sum * (3.0/(2.0**(len(r))-1.0))
            # convertim a flotant entre -1 i 3
            # amb r dígits decimals
    # Avaluem el cromosoma
    y = x * math.sin(10*math.pi*(x))+1.0
    return y
```

**Algorismes de cerca: cerca amb algorismes genètics.**

Exemple: Optimització d'una funció multimodal

Per fer la **mutació** imposem una probabilitat de mutació $p_m$ = 0.01 per a cada bit.

Per exemple, si tenim el cromosoma

v3 = (1110000000111111000101)

i seleccionem el cinquè bit per mutar, obtindrem

v3' = (1110100000111111000101).

Aquest cromosoma representa el valor x3'=1.721638, i per tant *f*(x3') = 2.343555, que s'ha incrementat respecte *f*(x3) = 2.250650.

**Algorismes de cerca: cerca amb algorismes genètics.**

Exemple: Optimització d'una funció multimodal

```
# Definim la mutació amb probabilitat mutprob
def mutacio(r,mutprob):
    import random
    for i in range(len(r)):
        if random.random()<mutprob:
            if r[i]==0: r[i]=1
            else: r[i]=0
    return r
```

## Algorismes de cerca: cerca amb algorismes genètics.

Exemple: Optimització d'una funció multimodal

Per al **creuament** d’una parella de cromosomes escollirem aleatòriament un punt de tall i intercanviarem informació per crear els dos descendents.

```
# Definim el creuament
def creuament(r1,r2):
    import random
    i=random.randint(1,len(r1)-2)
    return r1[:i]+r2[i:],r1[i:]+r2[:i]
```

## Algorismes de cerca: cerca amb algorismes genètics.

Exemple: Optimització d'una funció multimodal

Si iterem l'algorisme 150 generacions trobem que el millor cromosoma és $v_{max}$ = (1111001101000100000101), que correspon al valor $x_{max}$ = 1.850773.

L'evolució de l'algorisme es pot avaluar a partir del millor valor de la funció aconseguit en cada generació.

| Generació | Avaluació de f |
|---|---|
| 1 | 1.441942 |
| 6 | 2.150003 |
| 8 | 2.250283 |
| 9 | 2.250284 |
| 10 | 2.250363 |
| 12 | 2.328077 |
| 39 | 2.344251 |
| 51 | 2.733893 |
| 99 | 2.849246 |
| 137 | 2.850217 |
| **145** | **2.850227** |

## **Algorismes de cerca: cerca amb algorismes genètics.**

Exemple: Optimització d'una funció multimodal

*Observació vàlida per a qualsevol problema de recerca amb algorismes genètics:*

**Quin és el factor que més pesa en el càlcul de la complexitat computacional de l'algorisme? El nombre d'avaluacions!**

En el nostre problema hem fet 50x150=7.500 avaluacions!

Els operadors genètics tenen un cost computacional nul, per tant la complexitat de l'algorisme és el nombre d'avaluacions per la complexitat de l'avaluació.

## Annex: Truc de representació binària

En el cas concret de representar en forma binària nombre naturals, s'ha vist que hi ha una codificació que per alguns problemes funciona millor que la clàssica: el **codis de Gray.**

Aquests codis representen cada nombre de la seqüència $0 \ldots 2^N$ com una seqüència de bits de longitud $N$ tal que dos sencers adjacents tenen una representació que difereix només en un bit. D'això se’n diu la propietat d'**adjacència**.

**Codi Binari:** ***(000, 001, 010, 011, 100, 101, 110, 111)***

**Codis de Gray** ***(000, 001, 011, 010, 110, 111, 101, 100).***