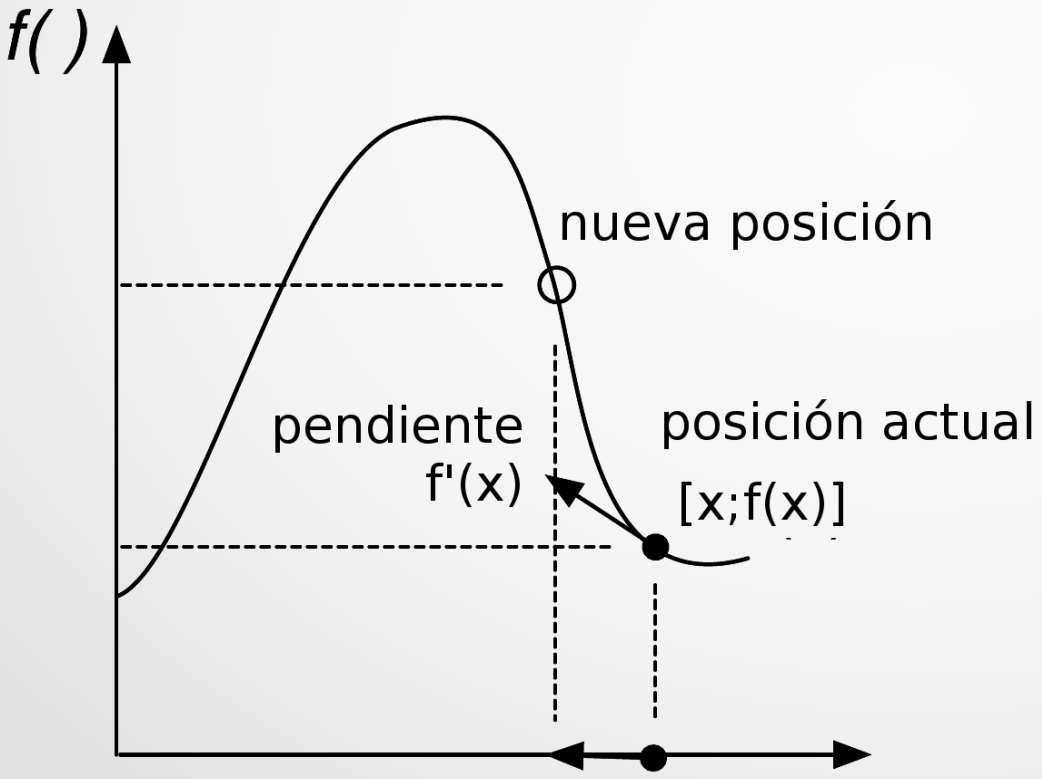
Análisis de Datos 2

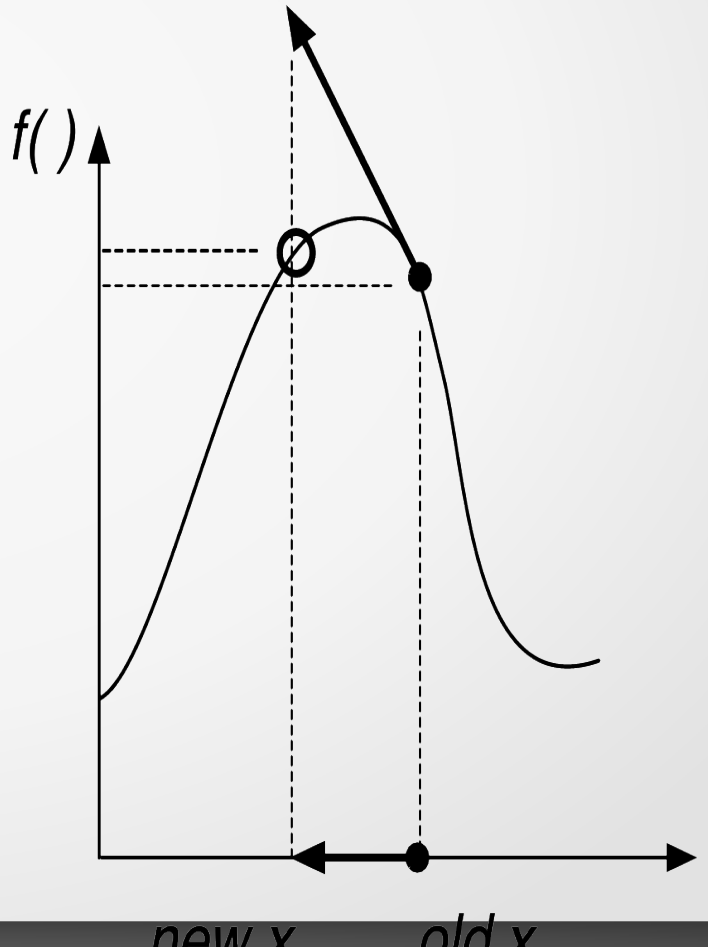
Búsqueda local
en algoritmos genéticos

Gradiente

$$x \leftarrow x + \alpha \cdot f'(x)$$

$$x \leftarrow x - \alpha \cdot \frac{f'(x)}{f''(x)}$$





Escalada voraz

- Iterar hasta que no mejore:

  nueva S ← arg máx {f(S') | S' vecina de S}

3	4	4	4	5	6	6	5	4	4
4	4	4	5	6	7	8	9	9	5
4	5	5	5	6	7	9	10	9	5
4	5	5	6	6	7	9	9	9	6
4	5	4	5	6	6	7	7	6	6
4	4	4	4	5	6	6	6	6	5
3	3	4	4	5	5	5	5	5	4

Escalada voraz

- Iterar hasta que no mejore:

  nueva S ← arg máx {f(S') | S' vecina de S}

3	4	4	4	5	6	6	5	4	4
4	4	4	5	6	7	8	9	9	5
4	5	5	5	6	7	9	10	9	5
4	5	5	6	**6**	7	9	9	9	6
4	5	4	5	6	6	7	7	6	6
4	4	4	4	5	6	6	6	6	5
3	3	4	4	5	5	5	5	5	4

Escalada voraz

- Iterar hasta que no mejore:

  nueva S ← arg máx {f(S') | S' vecina de S}

3	4	4	4	5	6	6	5	4	4
4	4	4	5	6	7	8	9	9	5
4	5	5	5	6	7	9	10	9	5
4	5	5	6	6	**7**	9	9	9	6
4	5	4	5	6	6	7	7	6	6
4	4	4	4	5	6	6	6	6	5
3	3	4	4	5	5	5	5	5	4

Escalada voraz

- Iterar hasta que no mejore:

  nueva S ← arg máx {f(S') | S' vecina de S}

3	4	4	4	5	6	6	5	4	4
4	4	4	5	6	7	8	9	9	5
4	5	5	5	6	7	9	10	9	5
4	5	5	6	6	7	**9**	9	9	6
4	5	4	5	6	6	7	7	6	6
4	4	4	4	5	6	6	6	6	5
3	3	4	4	5	5	5	5	5	4

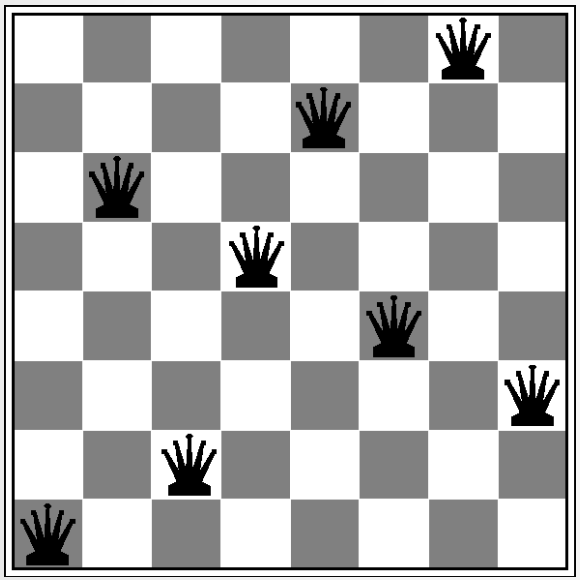
Escalada voraz

- Sea S la mejor solución actual
- mejora ← verdadero
- Mientras mejora = verdadero :
  - mejora ← falso
  - Para todo S’ en vecindario V(S)
    - Si S’ es mejor que S
      - S ← S’
      - mejora ← verdadero

Escalada voraz

- Ejemplo: n-reinas con parametrización con reposición

1 6 2 5 7 4 8 3



1 colisión

17 colisiones

4 3 2 5 4 3 2 3

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

vecindario: posiciones resultantes
de mover una reina en su columna

Escalada estocástica

- Sea S la mejor solución actual
- mejora ← verdadero
- Mientras mejora = verdadero :
  - mejora ← falso
  - C ← {S’ | S’ en V(S), S’ es mejor que S}
  - Si C no es vacío
    - S ← S’ aleatoriamente escogido de C
    - mejora ← verdadero

Escalada de primera opción

- Sea S la mejor solución actual
- mejora ← verdadero
- Mientras mejora = verdadero :
  - mejora ← falso
  - Bucle hasta visitar a lo sumo máxVecinos
    - Sea S’ elegido al azar de V(S)
    - Si S’ es mejor que S
      - S ← S’ ; mejora ← verdadero ; salir de Bucle
    - Si no
      - S’ ← al azar de V(S) ; ir a Bucle

Escalada de vecindarios variables

3	4	4	4	5	6	6	5	4	4
4	4	4	5	6	7	8	9	9	5
4	5	5	5	6	7	9	10	9	5
4	5	5	**6**	6	7	9	9	9	6
4	5	4	5	6	6	7	7	6	6
4	4	4	4	5	6	6	6	6	5
3	3	4	4	5	5	5	5	5	4

Escalada de vecindarios variables

3	4	4	4	5	6	6	5	4	4
4	4	4	5	6	7	8	9	9	5
4	5	5	5	6	7	9	10	9	5
4	5	5	**6**	6	7	9	9	9	6
4	5	4	5	6	6	7	7	6	6
4	4	4	4	5	6	6	6	6	5
3	3	4	4	5	5	5	5	5	4

Escalada de vecindarios variables

3	4	4	4	5	6	6	5	4	4
4	4	4	5	6	7	8	9	9	5
4	5	5	5	6	7	9	10	9	5
4	5	5	**6**	6	7	9	9	9	6
4	5	4	5	6	6	7	7	6	6
4	4	4	4	5	6	6	6	6	5
3	3	4	4	5	5	5	5	5	4

Escalada de vecindarios variables

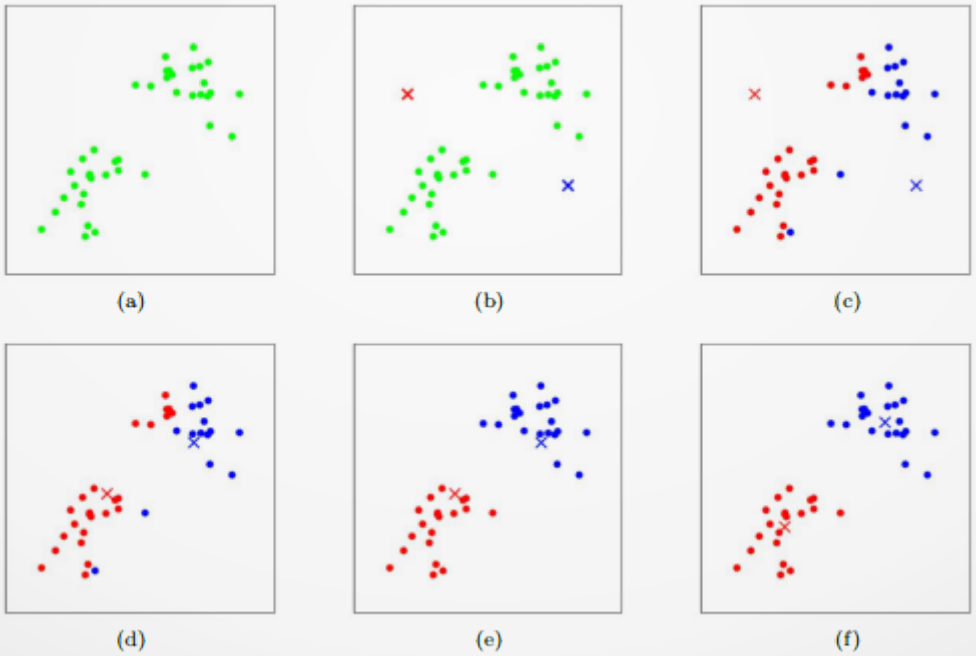
- Dada S solución y Vt lista de vecindarios (t = 1, …, T)
- mejora ← verdadero
- Mientras mejora = verdadero
  - mejora ← falso
  - Bucle para t = 1, …, T
    - S' ← busca local (S, Vt)
    - Si S' mejor que S entonces
      - S ← S'
      - mejora ← verdadero
      - salir de Bucle ; ir a Mientras

Escalada con perturbaciones

- Dada S la mejor solución actualmente
- Repetir hasta criterio de parada :
  - S' ← perturbar (S, K)
  - S' ← escalada (S')
  - Si S' mejor que S entonces S ← S'

Aplicación de perturbaciones: *k*-medias

- Partición de la muestra: z_{ip} = 0/1 según si *i* pertenece a *p*
- Muestra x_{ij} (individuo i, variable j); m_{pj} = media de clase p



Aplicación de perturbaciones: *k*-medias

- Algoritmo *k*-medias
  - mejora ← verdadero
  - Mientras mejora = verdadero
    - Calcular m_{ij}
    - Para todo *i*
      - Sea *p* tal que $z_{ip} = 1$
      - Sea d_{ip} = distancia (x_{ip}, m_{ip})
      - Si existe *p′* tal que $d_{ip'} < d_{ip}$
        - mejora ← verdadero
        - Modificar $z_{ip} \leftarrow 0$ y $z_{ip'} \leftarrow 1$

k-medias: perturbación básica

- Repetir *k* veces
  - Elegir aleatoriamente un individuo *i*
  - Elegir aleatoriamente una clase *p'*
  - Hacer $z_{ip'} := 1$ y $z_{ip} = 0$ para todo $p \neq p'$
- Esta perturbación tiene poco efecto

k-medias: perturbación „partir“

- Repetir *k* veces
  - Escoger aleatoriamente una clase p_1
  - Desplazar todos los individuos de p_1 a la clase más cercana → desaparece p_1
  - Escoger aleatoriamente una clase p_2
  - Asignar los individuos de p_2 aleatoriamente a p_1 o p_2
  - Aplicar *k*-medias restringido a p_1 y p_2

k-medias: perturbación „mezclar“

- Repetir *k* veces
  - Elegir aleatoriamente una clase p_1
  - Elegir aleatoriamente un individuo *i* de p_1
  - Sea p_2 la clase más próxima de *i* salvo p_1
  - Asignar aleatoriamente los individuos de p_1 y p_2 a p_1 o p_2
  - Aplicar *k*-medias restringido a p_1 y p_2

Mochila multidimensional 0/1

- Alonso, Caro, Montaña (2005)
- *n* objetos; *m* recursos (capacidad, peso); restricción *i* es b_i
- Objeto *j* produce p_j y consume a_{ij} del recurso *i*
- Maximizar $P'.X$ sujeto a que $A.X \leq B$
- $X = (x_j)_j$; x_j pertenece a {0, 1}

- La versión *LP-relajada* del problema supone que

  x_j pertenece a [0, 1]

Multiplicadores surrogados

- $w_j > 0$ (o, igualmente, $0 < w_j \leq 1$) ; $W = (w_j)_j$
- Maximizar $P'.X$ sujeto a que $W'.A.X \leq W'.B$
- $X = (x_j)_j$; x_j pertenece a [0, 1]

- Para todo W, $P'.X^* \leq P'.X_w^*$ donde
  - X^* es la mejor solución del LP-relajado
  - X_w^* es la mejor solución del LP con surrogados
- Buscar W que minimice $P.X_w^*$

A.G. para los multiplicadores

- Representación binaria de cada W con precisión q
- Antiadaptación $f(W) = P'.X_w^*$
- X_w^* puede obtenerse con algoritmo voraz para el problema de la mochila unidimensional LP-relajado

A.G. para mochila multidimensional

- Representación binaria para X (natural)
- Adaptación $f(X) = P'.X$
- Busca sólo en el espacio factible $A.X \leq B$
- Reparación de individuos infactibles mediante multiplicadores surrogados
- Mejora mediante búsqueda local

Reparador

- Ratios de utilidad de variables X dados multiplicadores W

  $(g_j)_j = W'.A$
  $u_j = p_j : g_j$
- Sea $X = (x_j)_j$ una solución no factible
- Sea φ permutación tal que $u_{\varphi(1)} \leq u_{\varphi(2)} \leq ... \leq u_{\varphi(n)}$
- Sea j:=1
- Mientras X sea no factible
  - Si $x_{\varphi(j)} = 1$ entonces $x_{\varphi(j)} := 0$
  - $j := j + 1$

Mejorador

- Ratios de utilidad de variables X dados multiplicadores W

  $(g_j)_j = W'.A$
  $u_j = p_j : g_j$
- Sea $X = (x_j)_j$ una solución factible
- Sea φ permutación tal que $u_{\varphi(1)} \geq u_{\varphi(2)} \geq \ldots \geq u_{\varphi(n)}$
- Iterar desde j:=1 hasta j:=n
  - Si $x_{\varphi(j)} = 0$ entonces $x_{\varphi(j)} := 1$
  - Si X es infactible, entonces $x_{\varphi(j)} := 0$

Operadores genéticos

- Selección mediante ruleta
- Entrecruzamiento uniforme
- Mutación con probabilidad p_M=0'1

Búsqueda local

- Sea X solución factible, sea W vector de multiplicadores
- Reparador[*j*] es como reparador pero sin modificar locus *j*
- Mejorador[*j*] es como mejorador pero sin modificar locus *j*
- Paseo aleatorio en el hipercubo $\{0, 1\}^n$ hasta no mejora
- Se conmuta x_j sii hay una mejora de la adaptación
- Si se conmuta entonces
  - Si X factible, aplicar Mejorador[j]
  - Si X no factible, aplicar Reparador[j] y Mejorador

Búsqueda local

- mejora ← verdadero
- Mientras mejora = verdadero
  - mejora ← falso
  - P ← permutación de 1 a n (paseo aleatorio)
  - Desde j ← 1 hasta j ← n
    - X' ← X
    - Conmutador[$x'_{p(j)}$]
    - Si f(X) ≥ f(X') entonces
      - X ← X'
      - mejora ← verdadero

Búsqueda local

- Conmutador[$x_{p(j)}$]:
  - Si $x_{p(j)} = 1$ entonces
    - $x_{p(j)} \leftarrow 0$
    - X ← Mejorador[p(j)] (X)
  - Si $x_{p(j)} = 0$ entonces
    - $x_{p(j)} \leftarrow 1$
    - X ← Reparador[p(j)] (X)
    - X ← Mejorador (X)

Búsqueda local

- Se aplica sólo cada *t* generaciones
- A todos los individuos de la población
- Para mantener diversidad en población P(*k.t*), *k* = 1, 2, 3...:
  - P(*k.t*+1) := {}
  - Para cada *X* en P(*k.t*) hacer
    - *Y* := busca local (*X*)
    - Si *Y* pertenece a P(*k.t*) o P(*k.t*+1), añadir *X* a P(*k.t*+1)
    - Si no, añadir *Y* a P(*k.t*+1)

Búsqueda local

- Generación inicial
  - Iterar
    - Generar permutación *P* aleatoria de orden *n*
    - Aplicar Mejorador a (0, …, 0) según *P*

Sistemas de ecuaciones de palabras

- Alonso, Drubi, Gómez-García, Montaña (2004)
- Aplicación: hallar patrones en cadenas
- xx01x1y = 01001010100011010101010 → fácil
- x01x1y = 1y0xy → difícil

Sistemas de ecuaciones de palabras

- A = alfabeto de constantes; Ω = alfabeto de variables
- A* = conjunto de palabras formadas por A
- w = elemento de A*; |w| = longitud de w
- d-WES = sistema de ecuaciones de palabras con

  {longitud de toda solución} ≤ d

Codificación

- $A = \{0,1\}$
- $\Omega = \{x_1, \ldots, x_m\}$
- $S = \{L_1 = R_1, \ldots, L_n = R_n\}$
- $|\sigma(x_i)| \leq d$
- Individuo = m cromosomas
- Cromosoma = palabra binaria de longitud ≤ d

Antiadaptación

- Extensión de la distancia de Hamming a palabras de distintas longitudes
- $H(a,b) = \text{máx}\{|a|, |b|\} - \#\{k \mid 1 \leq k \leq \text{mín}\{|a|, |b|\}, a[k]=b[k]\}$
- Antiadaptación = $f(a) = \Sigma\, H(L_i(a), R_i(a))$
- Distingue entre individuos que satisfacen el mismo número de ecuaciones
- Existe solución del d-WES si existe „a“ tal que $f(a) = 0$

Operadores genéticos

- Selección: ruleta
- Mutación: cada locus es invertido con probabilidad 1:d
- Entrecruce:
  - Por cromosomas homólogos (i = 1, …, n)
  - Uniforme (UX) entre partes comunes a la izquierda
  - Un trozo aleatorio del resto
  - $k = |a_i| < |b_i|$; $k' = \text{mín}\ \{|b_i|, \text{azar}\ \{k+1, \ldots, d\}\}$

    → cruce $(a_i, b_i) = UX\ (a_i, b_i[1..k]) \cup b_i[k+1..k']$

Búsqueda local

- Sea el k-vecindario de la solución „a“

  $U_k(a) = \{b \mid \text{máx}_i\ H(a_i, b_i) \leq k\}$

- Búsqueda local 1 (LS1)
  - p = probabilidad de ruido
  - Con Pr = p elegir un „b“ al azar de $U_1(a)$
  - Con Pr = 1 – p elegir el mejor adaptado de $U_1(a)$
  - Iterar
    - Mientras no se obtenga una solución „b“ con f(b)=0
    - Hasta un número máximo M de iteraciones

Búsqueda local 2 (LS2)

- Paseo aleatorio en $U_k(a)$; $k = \Sigma_{1 \le i \le m} |a_i|$
- Sea H el conjunto de locus de a
- Repetir hasta que H esté vacío
  - Sea g = (i,j) en H al azar
  - Sea $V_g(a)$ subconjunto de $U_1(a)$ tal que sus elementos
    - se obtienen de conmutar el locus „g“ de „a“ o
    - añadiendo/borrando el último locus del i-ésimo cromosoma de „a“ (sólo si $j=|a_i|$)
  - Buscar a' = óptimo en $V_g(a)$; si a'=a, FIN; si no, $a \leftarrow a'$
  - Quitar g de H

Bibliografía

- Siarry 2014. Métaheuristiques. Ed. Eyrolles
- Alonso, Drubi, Gómez-García, Montaña 2004. Advances in A.I. - S.B.I.A.