Control Inteligente

Fundamentos de Algoritmos Evolutivos
L11

Luis Moreno, Santiago Garrido, Dorin Copaci

Dpto. Ing. de Sistemas y Automática
Universidad Carlos III
Madrid

Oct 2019

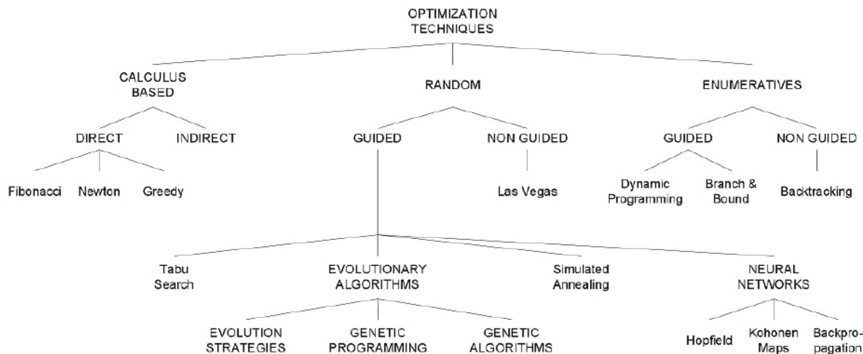
Table of contents

1 Algoritmos Evolutivos

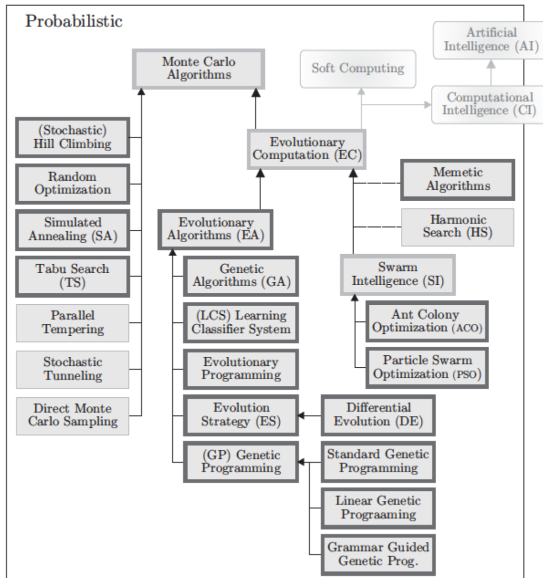
Introducción

- Detrás de todo algoritmo con capacidad de aprendizaje o de adaptación está siempre un algoritmo de optimización.
- Existen diferentes clasificaciones o axonomías de los algoritmos de optimización.

Taxonomías de los algoritmos de optimización.



- Otra clasificación los divide en deterministas y estocásticos o probabilistas



Probabilistic
Artificial Intelligence (AI)
Monte Carlo Algorithms
Soft Computing
Computational Intelligence (CI)
(Stochastic) Hill Climbing
Evolutionary Computation (EC)
Random Optimization
Memetic Algorithms
Simulated Annealing (SA)
Evolutionary Algorithms (EA)
Harmonic Search (HS)
Tabu Search (TS)
Genetic Algorithms (GA)
Swarm Intelligence (SI)
Parallel Tempering
(LCS) Learning Classifier System
Ant Colony Optimization (ACO)
Stochastic Tunneling
Evolutionary Programming
Particle Swarm Optimization (PSO)
Direct Monte Carlo Sampling
Evolution Strategy (ES)
Differential Evolution (DE)
(GP) Genetic Programming
Standard Genetic Programming
Linear Genetic Prograaming
Grammar Guided Genetic Prog.

Métodos

En nuestro caso haremos una separación en dos grandes grupos:

- Métodos mono-punto.
  Existe un amplia variedad de métodos que exploran el espacio de posibles soluciones mediante la evolución controlada por algún criterio de un único punto que va siendo desplazado hasta alcanzar el mínimo de la función de coste que se explora.
- Métodos multi-punto o basados en poblaciones.
  Estos métodos utilizan múltiples puntos que exploran en paralelo el espacio de soluciones para buscar el mínimo de la misma.

Métodos de búsqueda mono-punto

Métodos de búsqueda mono punto

- Existe un amplia variedad de métodos desarrollados para resolver el problema de la minimización de funciones.
- Estos métodos pueden ser agrupados en dos grandes familias dependiendo de si están basados en la derivada o no.

Métodos basados en la derivada

- Los métodos basados en la derivada constituyen la aproximación clásica al problema de optimización. La idea básica de estos métodos puede formularse del siguiente modo. Se expande una función arbitraria $f(\mathbf{x})$, donde $\mathbf{x} = (x_0, x_1, \ldots, x_n)^T$, en una serie de Taylor

$$f(\mathbf{x}) \approx h(\mathbf{x}) = f(\mathbf{x}_0) + \frac{\nabla f(\mathbf{x}_0)}{1!}.(\mathbf{x} - \mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T.\frac{\nabla^2 f(\mathbf{x}_0)}{2!}.(\mathbf{x} - \mathbf{x}_0) + \ldots \tag{1}$$

  donde $\mathbf{x}_0$ es el punto alrededor del cual la función f es expandida, ∇f es el vector gradiente de f y $\nabla^2 f$ es la matriz Hessiana de f.
- La función $h(\mathbf{x})$, considerando el desarrollo hasta segundo orden, es una función cuadrática, cuyo mínimo $\mathbf{x}_{min}$ puede obtenerse resolviendo $\nabla h(\mathbf{x}) = 0$.

Métodos basados en la derivada

- Obteniendo el gradiente de la función aproximada $h(\mathbf{x})$ tenemos que

$$\nabla h(\mathbf{x}) = \nabla f(\mathbf{x}_0) + \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \tag{2}$$

- y como en el mínimo $\nabla h(\mathbf{x}_{min}) = 0$ tendremos que

$$\nabla f(\mathbf{x}_0) + \nabla^2 f(\mathbf{x}_0).(\mathbf{x}_{min} - \mathbf{x}_0) = 0 \tag{3}$$

- y de esta expresión obtenemos que

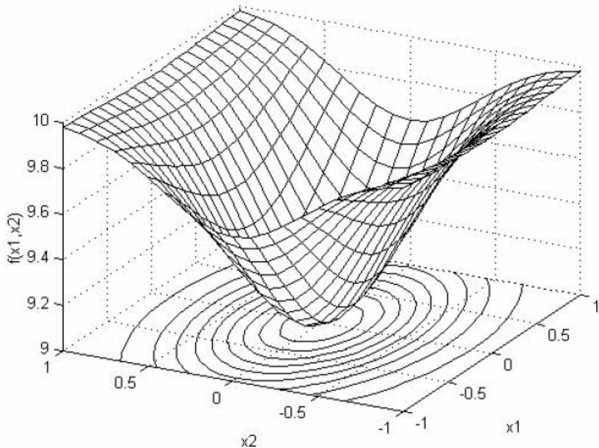
$$\mathbf{x}_{min} = \mathbf{x}_0 - (\nabla^2 f(\mathbf{x}_0))^{-1}.\nabla f(\mathbf{x}_0) \tag{4}$$

Métodos basados en la derivada

- Los métodos basados en la derivada son extremadamente potentes en aquellas aplicaciones donde la función objetivo es cuadrática. En estos casos, la ecuación 4 se aplica iterativamente para obtener el mínimo de la función objetivo, desafortunadamente la mayoría de los problemas de optimización carecen de esta favorable propiedad.
- Para que sean efectivos, este grupo de métodos requieren algunas propiedades en la función objetivo: debe de ser *diferenciable y uni-modal*, es decir, la función debe de tener un único mínimo.

Métodos basados en la derivada

- Función objetivo derivable y uni-modal



Métodos basados en la derivada

- Entre estos métodos tenemos:
  - Máxima pendiente (Steepest descent)
  - Newton-Raphson
  - Quasi-Newton
  - Gradiente conjugado (Conjugate gradient)

Máxima pendiente

- El método de máxima pendiente es el más simple de los métodos basados en el gradiente para encontrar el mínimo de una función (diferenciable y uni-modal).
- Para una función arbitraria $f(\mathbf{x})$, donde $\mathbf{x} = (x_0, x_1, \ldots, x_n)^T$, el método de máxima pendiente (steepest descent method) busca el próximo punto de acuerdo a la siguiente expresión

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \cdot \nabla f(\mathbf{x}_k) \tag{5}$$

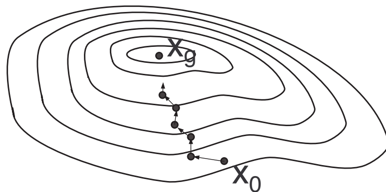
  donde $\nabla f(\mathbf{x}_k)$ expresa el valor del vector gradiente de f en el punto $\mathbf{x}_k$.
- Esta expresión puede obtenerse directamente a partir de 4 reemplazando $\nabla^2 f(\mathbf{x}_k)$ por la matriz identidad.
- Dado que el gradiente es negativo, los puntos función abajo $\mathbf{x}_{k+1}$ estarán más próximos al mínimo que $\mathbf{x}_k$ a menos que el paso sea demasiado grande.

Máxima pendiente

- El término $\mathbf{x}_0$ es la hipótesis inicial. El factor γ_k ($\gamma_k > 0$) nos permite tener control sobre el tamaño del paso de avance usado, y puede ser especificado a priori (a menudo es una constante $\gamma_k = \gamma$) o se elige iteración-a-iteración en base a una solución a

$$min_{\gamma \geq 0}\{f(\mathbf{x}_k - \gamma \cdot \nabla f(\mathbf{x}_k))\}. \tag{6}$$

- Este problema secundario de optimización se denomina búsqueda lineal (*line search*).



Máxima pendiente

- Este método clásico nos permite ver la existencia de un *problema de determinación del tamaño del paso de avance* (step size).
- El papel del factor γ_k es regular la longitud del paso de avance que da el algoritmo. Si es demasiado grande o demasiado pequeño puede llegar a dificultar que el algoritmo converja al valor correcto $\mathbf{x}^*$, incluso aunque los pasos se den en las direcciones correctas.
- A pesar de las ventajas que desde un punto de vista formal de la convergencia del algoritmo a la solución correcta, es un método relativamente ineficiente en comparación con otros métodos basados en el gradiente.
- Además es muy sensible a las transformaciones y cambios de escala.

Newton-Raphson

- El método de Newton-Raphson se basa en la ecuación básica de recursión 5

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k . \nabla f(\mathbf{x}_k) \tag{7}$$

  en la que se introduce un escalado vía la inversa de la matriz Hessiana. Esto tiene la ventaja potencial de acelerar la convergencia de forma muy significativa, pero tiene el inconveniente de que puede ocasionar que el algoritmo sea más inestable

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla^2 f(\mathbf{x}_k)^{-1} . \nabla f(\mathbf{x}_k) \tag{8}$$

- Si la función de coste $f()$ es una *función cuadrática*, el método de Newton-Raphson **convergerá en un sólo paso** desde cualquier punto de partida (ningún otro algoritmo puede converger más rápido). En la práctica, ésta no es una situación muy común ya que pocas funciones de coste son funciones cuadráticas.

Newton-Raphson

- Pese a todo, en las proximidades de la solución $\mathbf{x}^*$, $f()$ será casi cuadrática para cualquier función dos veces diferenciable por lo que se puede esperar que el método de Newton-Raphson sea rápido una vez que esté cerca del mínimo $\mathbf{x}^*$.
- Para funciones de coste no lineales de tipo general (no-cuadráticas), el Hessiano $\nabla^2 f$ puede no ser definido positivo, lo que puede ocasionar el estancamiento o la divergencia.
- No hay garantìa de que $f(\mathbf{x}_{k+1}) \leqslant f(\mathbf{x}_k)$.
- A diferencia del método de máxima pendiente, el de Newton-Raphson es **invariante a transformaciones** y no se ve afectado por grandes diferencias en el escalado en los elementos de $\mathbf{x}$.
- Si denominamos $d_k = -\nabla^2 f(\mathbf{x}_k)^{-1}.\nabla f(\mathbf{x}_k)$ el tamaño del paso puede optimizarse si se hace una búsqueda lineal de $f(\mathbf{x}_k + \alpha.d_k)$ para encontrar un valor óptimo del tamaño de paso α.

Quasi-Newton

- Esta familia de métodos aproxima, en la ec 4, la matriz $\nabla^2 f(\mathbf{x}_0)$ por un mecanismo más elaborado que en el método del máximo gradiente.
- En este grupo encontramos los métodos de: Gauss-Newton, Fletcher-Reeves, Davidon-Fetcher-Powell y Levenberg-Marquard.
- Este grupo de métodos aproxima la inversa de la matriz Hessiana por esquemas que requieren diferentes cálculos con matrices.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\nabla^2 f(\mathbf{x}_k))^{-1} \cdot \nabla f(\mathbf{x}_k) \tag{9}$$

Gradiente conjugado

- El método del gradiente conjugado (conjugate gradient) usa una optimización lineal en las direcciones conjugadas para evitar el cálculo de la segunda derivada que implica el Hessiano.
- Todos estos métodos requieren que la función objetivo sea diferenciable, y si usan funciones de tipo cuadrático, exhiben una capacidad de convergencia extremadamente rápida.
- Esta característica no ocurre necesariamente si la función objetivo no es cuadrática.

Métodos no derivativos (direct search)

- Los métodos derivativos presentan problemas ya que muchas funciones no son diferenciables, y en estos casos la ausencia de una derivada hace que sean necesarios métodos diferentes que permitan determinar el mínimo de la función objetivo sin usar la derivada.
- Los métodos no derivativos son conocidos como métodos de búsqueda directa (*direct search* methods) y están menos basados en el calculo diferencial que los métodos derivativos.
- Estas técnicas usan reglas heurísticas y saltos condicionales para explorar la función objetivo.
- La idea general de estos métodos consiste en generar un punto en el espacio de estados donde se busca la solución y comprobar el valor de la función objetivo en dicho punto, es decir es un esquema de generación-comprobación de posibles soluciones.

Métodos no derivativos (direct search)

- Entre estos métodos tenemos:
  - Búsqueda por fuerza bruta (brute force search)
  - Paseo aleatorio (Random walk)
  - Hooke-Jeeves
  - Simulated Annealing

Método de fuerza bruta (Brute force search)

- Los métodos de búsqueda directa comienzan con un punto de arranque inicial.
- En el caso particular del método de fuerza bruta, el punto seleccionado sigue un patrón de exploración retículado pre establecido que cubre una región acotada.
- Se almacena el mejor de los resultados obtenidos hasta cada momento (Fig. 1).

Método de fuerza bruta

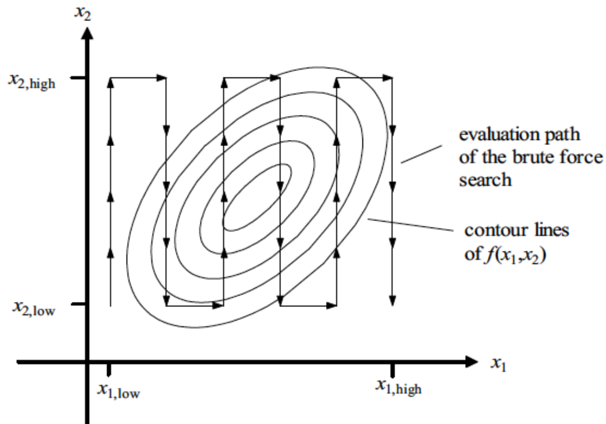


Figura: Brute force search trajectory.

Método de fuerza bruta

- Este método presenta un problema de tamaño del paso de avance, la razón se debe a que el tamaño de la retícula es definido a priori y no tiene en cuenta la función objetivo que se está evaluando.
- Si el tamaño de la retícula (grid size) es pequeño el coste computacional explota ($O(N^D)$, donde N es el número de puntos en cada dimensión, y D es el número de dimensiones)
- Si el tamaño de la retícula es grande es posible que no encontremos el mínimo verdadero.
- Obviamente, estos métodos enumerativos son muy escasamente utilizados por su alto coste computacional.

Paseo aleatorio (Random walk)

- El método del paseo aleatorio (Random walk) se debe a Gross y Harris (1985), introduce un muestreo aleatorio para generar nuevos puntos donde evaluar la función de coste objetivo. .
- Los nuevos puntos son generados mediante a adición de una perturbación aleatoria $\Delta(\mathbf{x})$, al punto de búsqueda actual $\mathbf{x}_k$.
- La perturbación aleatoria en cada dimensión $\Delta(x_i)$ sigue una distribución Gaussiana.
- El criterio de selección en el paseo aleatorio es extremadamente simple, se acepta el nuevo punto de ensayo si el valor de su función de coste es menor que el valor de la función de coste en el punto base, en otro caso el punto antiguo se mantiene y es generado un nuevo punto tentativo.
- En la figura se muestra el comportamiento de este método 2 .

Paseo aleatorio (Random walk)

- Al igual que en el método anterior, la técnica del paseo aleatorio no tiene una forma adecuada de ajustar el paso de avance del algoritmo (step size), porque la desviación standard usada en la generación de la perturbación aleatoria Gaussiana es, por lo general, mal conocida.

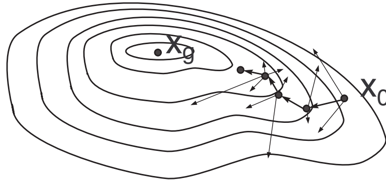


Figura: Paseo aleatorio.

Hooke-Jeeves

- El método de Hooke-Jeeves (pattern search method) es un método de búsqueda directa mono-punto que tiene en cuenta el problema del tamaño del paso de avance (Hooke yJeeves 1961, Pierre 1986, Schwefel 1994).
- La idea básica de este método consiste en explorar en cada eje de coordenadas. En cada punto de prueba, y con un paso de avance determinado, se exploran todas las D dimensiones en las direcciones positivas y negativas.
- La función objetivo en dichos puntos es comparada y se determina el mejor de los puntos probados. Si el mejor de los puntos probados es mejor que el punto base previo $\mathbf{x}_i$, entonces se hace un intento de movimiento en la misma dirección.
- Si el mejor de los puntos probados es peor que el punto base previo, el paso de avance ha sido demasiado grande y se repite el procedimiento con un paso de avance menor (se usa un factor F de reducción).

Hooke-Jeeves Method

while $(h > h_{min})$ $\{$
  for $(i = 0; i < D; i++)$ $\{$
    if $(f(\mathbf{x}_i + \mathbf{e}_i * \mathbf{h}) < f(\mathbf{x}_i))$
      $\mathbf{x}_i = \mathbf{x}_i + \mathbf{e}_i * \mathbf{h};$
    elseif $(f(\mathbf{x}_i - \mathbf{e}_i * \mathbf{h}) < f(\mathbf{x}_i))$
      $\mathbf{x}_i = \mathbf{x}_i - \mathbf{e}_i * \mathbf{h};$
  $\}$
  if $(f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i))$
    $\mathbf{x}_{i+2} = \mathbf{x}_{i+1} + (\mathbf{x}_{i+1} - \mathbf{x}_i);$
    if $(f(\mathbf{x}_{i+2}) < f(\mathbf{x}_{i+1}))$
      $\mathbf{x}_i = \mathbf{x}_{i+2};$
    else
      $\mathbf{x}_i = \mathbf{x}_{i+1};$
  else
    $h = h * F;$
$\}$

Hooke-Jeeves

- El método de reducción del paso de avance que implemanta el algoritmo de Hooke-Jeeves lo hace más eficiente que los métodos de fuerza bruta y paseos aleatorios.
- Pero debido a que el tamaño del paso nunca es incrementado existe una alta posibilidad de quedar atrapado en un mínimo local.

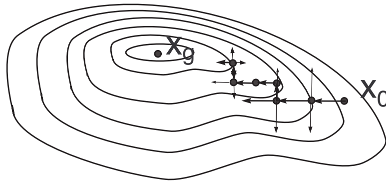


Figura: Hooke-Jeeves.

Optimización Local versus Optimización Global

- Cuando existe más de un mínimo local, es decir en funciones multi-modales aparece el problema del punto de partida.

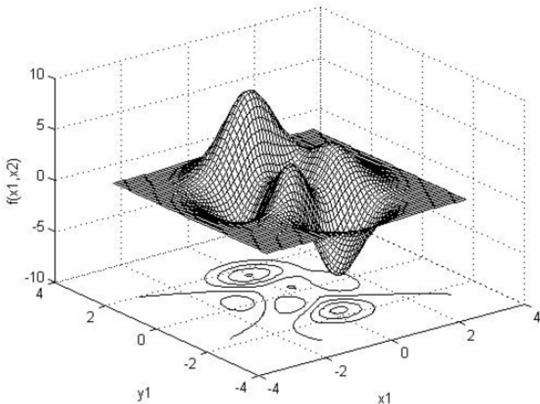


Figura: Función con mínimos locales.

Optimización Local versus Optimización Global

- El problema del punto de partida se refiere a la tendencia de un cierto método de optimización con un criterio de selección de tipo greedy a encontrar sólo el mínimo de de la cuenca de atracción en la que fue inicializado.
- Este mínimo puede no ser el mínimo global, por lo que muestrear una función multimodal en la vecindad de un óptimo global es esencial evitar este problema.
- Dado que la distribución Gaussiana no está acotada, existe una probabilidad finita de que el paseo aleatorio pueda genera un punto nuevo y mejor en una cuenca de atracción distinta de la que contiene al punto base actual. En la práctica estos saltos inter-cuencas con éxito tienden a ser muy escasos.
- Un método que incrementa la posibilidad de que un punto salte a otra cuenca de atracción es el método del Simulated Annealing (SA).

Simulated Annealing (SA)

- El método de Simulated Annealing (SA) fué propuesto por Kirkpatrick et al. en 1983 (Press et al. 1992), muestrea la superficie de la función objetivo mediante la modificación del criterio usado para aceptar algún movimiento hacia arriba mientras se continúa para aceptar todos los movimientos función de coste abajo.
- La probabilidad de aceptar un vector o punto tentativo que tiene mayor valor de la función de coste que el punto base actual decrece a medida que la diferencia entre los valores de sus funciones de coste aumenta.
- La probabilidad de aceptación también se decrementa con el número de evaluaciones de la función de coste (puntos explorados), es decir, después de un tiempo razonablemente largo, el criterio de selección se vuelve greedy (ávido, ansioso).

Simulated Annealing (SA)

- El método de paseo aleatorio ha sido usado tradicionalmente en conjunción con el método de Simulated Annealing para generar nuevos vectores o puntos a explorar, pero en principio cualquier método de búsqueda puede ser modificado e incorporar el esquema de selección SA.

Simulated Annealing (SA)

- El término annealing (recocido) se refiere al proceso de enfriamiento lento al que se somente a una substancia fundida para que sus átomos tengan la oportunidad de alcanzar una configuración de energía mínima.
- Si la substancia se mantiene cerca del equilibrio a temperatura T, entonces la energía de sus átomos, E, se distribuye de acuerdo con la ecuación de Boltzmann

$$P(E) \sim e^{\frac{E}{kT}} \tag{10}$$

  donde k es la constante de Boltzmann.
- Igualando la energía al valor de la función de coste, el algoritmo SA intenta explotar un proceso natural de minimización vía el algoritmo de Metrópolis (Metrópolis et al. 1953).

Simulated Annealing (SA)

- El algoritmo de Metrópolis implementa la ecuación de Boltzmann como una probabilidad de selección. Mientras que el movimiento es hacia valores menores de la función de coste estos se aceptan siempre, pero cuando los movimientos son hacia valores mayores de la función de coste estos son aceptados solamente si un número aleatorio generado según una distribución de probabilidad uniforme en el intervalo [0,1] es menor que el término exponencial:

$$\Theta = e^{\frac{-d.\beta}{T}} \tag{11}$$

- donde:
  - La variable, d, es la diferencia entre el valor de la función objetivo con coste creciente (uphill) y el de la función objetivo en el punto base actual, es decir, su diferencia de energía .
  - La variable, Θ, nos da la probabilidad de aceptacin que decrece a medida que d aumenta y/o que T decrece.
  - El valor de β es una variable de control que depende del problema y debe determinarse empíricamente (prueba y error).

Simulated Annealing (SA)

Simulated Annealing

$fbest = f(x_0)$; // start with some base point
$T = T_0$; // and some starting temperature
while (convergence criterion not yet met) {
  $\Delta x = generatedeviation()$; //e.g., a Gaussian distribution
  if $(f(x_0 + \Delta x) < f(x_0))$ {
    // if improvement can be made
    $fbest = f(x_0 + \Delta x)$;
    $x_0 = x_0 + \Delta x$; // new, improved base point }
  else {
    $d = f(x_0 + \Delta x) - f(x_0)$; //positive value
    $r = rand()$; //generate uniformly distr. variable ex $[0, 1]$
    if $(r < exp(-d * beta/T))$ {
      // Metropolis algorithm
      $x_0 = x_0 + \Delta x$; // new base point derived from uphill move }
  }
$T = T * reductionfactor$;
}

Simulated Annealing (SA)

- Uno de los inconvenientes de la técnica de Simulated Annealing consiste en el esfuerzo que debe dedicarse para encontrar un esquema de annealing que disminuya T a alta velocidad.
- Si T se reduce demasiado deprisa, el algoritmo se comportará como un optimizador local y quedará atrapado en la cuenca de atracción del mínimo local en el que comienze.
- Si T no disminuye suficientemente rápido, los cálculos se hacen muy costosos computacionalmente.

Simulated Annealing (SA)

- Se han realizado muchas mejoras sobre la versión estándar del algoritmo SA algorithm (Ingber 1993) y este algoritmo ha sido utilizado en lugar de criterios greedy en algoritmos de búsqueda directa como el método de Nelder-Mead (Press et al. 1992).
- El problema del tamaño del paso de avance persiste en este algoritmo, sin embargo, y posiblemente es la razón por la que este método es poco usado en la minimización de funciones continuas.
- Por el contrario, este método ha sido muy aplicado junto con cualquier método de búsqueda directa en optimización combinatoria (Syslo et al. 1983; Reeves 1993).

Métodos de búsqueda multi-punto

Métodos de búsqueda multi-punto

- La mayoría de los métodos mono-punto tienen tendencia a encontrar el mínimo de la función en la cuenca de atracción en la que se encuentran situados cuando han sido inicializados.
- Este mínimo, en el caso de funciones objetivo multi-modales puede ser un mínimo local. En la técnica del paseo aleatorio debido a que la perturbación es Gaussiana y no acotada, existe una probabilidad de encontrar el mínimo verdadero de la función.
- Pero esta probabilidad es extremadamente pequeña, y tiene tendencia a fallar si el mínimo verdadero está situado lejos del área de exploración actual.
- Una forma de tratar de resolver el problema de la localidad de los algoritmos mono-punto es usar métodos multi-punto o multi-start (multi-inicio).

Métodos multi-punto derivativos

Dentro de los métodos multi-punto basados en el uso de la derivada tenemos:

- Técnicas multi-start (multi-inicio)
- Métodos de Clustering (agrupamiento)

Técnicas multi-start (multi-inicio)

- Las técnicas multi-start (multi-inicio) usan puntos obtenidos mediante muestreo para iniciar el proceso de optimización desde diferentes puntos iniciales.
- Cada una de los puntos iniciales obtenidos por muestreo sirven como punto de arranque de un método derivativo mono-punto que hace una búsqueda local en la cuenca de atracción en la que comienza el punto muestreado (Boender and Romejin 1995).
- Esta técnica incrementa la probabilidad de encontrar el mínimo verdadero, pero aún tiene ciertos problemas. Por una parte requiere que la función objetivo sea diferenciable, ya que en caso de no serlo los métodos locales de tipo derivativo no pueden usarse.
- Por otra parte, resulta difícil de determinar a priori cuantos puntos de arranque o inicio son suficientes para encontrar el mínimo verdadero de la función.

Métodos de Clustering

- Los métodos de Clustering fueron propuestos por Torn and Zelinkas 1989 y Janka 1999, aplican técnicas de clustering para identificar los puntos de muestreo que corresponden a la misma cuenca de aracción (el mismo cluster).
- Estos métodos requieren que todos los puntos visitados previamente sean almacenados, lo cual en el caso de funciones altamente multi-modales puede ser muy costoso desde el punto de vista de los requisitos computacionales.
- Como resultado, los algoritmos de clustering se suelen limitar a problemas con un número pequeño de dimensiones.

Métodos multi-punto no-derivativos

Entre los métodos multi-punto no-derivativos, también denominados métodos basados en poblaciones (population-based methods), nos encontramos los siguientes:

- Método Nelder-Mead.
- Método de búsqueda aleatoria controlada (Controlled Random Searh, CRS).
- Los algoritmos evolutivos (Evolutionary algorithms ,EAs) constituyen un grupo muy amplio de métodos multi-punto no-derivativos, reciben este nombre debido a que imitan la evolución Darwinista. Entre estos tenemos:
  - Las estrategias d evolución (*Evolution strategies* , ESs) propuestas por Rechenberg (1973) y Schwefel (1994),
  - Los algoritmos genéticos (*Genetics algorithms*, GAs) propuestas por Holland (1962) y Goldberg (1989).
  - El método de Differential Evolution, DE, propuesto por Storn and Price (1999).
  - Los métodos de tipo PSO (particle swarm optimization)
  - Los métodos de tipo ACO (ant colonies).

Nelder-Mead

El método de búsqueda poliédrica de Nelder-Mead (Nelder-Mead, 1965) trata de resolver el problema del tamaño del paso de avance permitiendo que el tamaño de paso se expanda o contraiga según sea necesario. El algoritmo opera del siguiente modo:

- Se comienza formando un poliedro de dimensión $(D+1)$, o simplex, de $(D+1)$ puntos, $\mathbf{x}_i, i = 0, 1, \ldots, D$, que son distribuidos aleatoriamente por el espacio del problema (ej.; cuando $D = 2$, el simplex es un triángulo).
- Los puntos son ordenados en sentido ascendente del valor de la función objetivo, de forma que $\mathbf{x}_0$ es el mejor punto y $\mathbf{x}_D$ es el peor punto.

Nelder-Mead

- (cont).
  - Para obtener un nuevo punto de prueba, $\mathbf{x}_r$, el peor punto, $\mathbf{x}_D$, es reflejado a través del lado opuesto del poliedro usando un factor de ponderación, F_1:
  $$\mathbf{x}_r = \mathbf{x}_D + F_1(\mathbf{x}_m - \mathbf{x}_D) \quad (12)$$
  donde $\mathbf{x}_m$ es el centroide de la cara opuesta al punto $\mathbf{x}_D$;
  $$\mathbf{x}_m = \frac{1}{D}[\sum_{i=0}^{D-1} \mathbf{x}_i] \quad (13)$$
  - Si la reflexión a través del centroide mejora el valor de la función objetivo, $f(\mathbf{x}_m) < f(\mathbf{x}_0)$, entonces el algoritmo da otro paso en la misma dirección basándose en la hipótesis de que una mejora adicional en esta dirección es todavía posible. Este segundo punto a explorar se obtiene como una expansión del punto previo con un segundo factor de escala F_2,
  $$\mathbf{x}_e = \mathbf{x}_r + F_2(\mathbf{x}_m - \mathbf{x}_D) \quad (14)$$

Nelder-Mead

- (cont).
  - Si este punto de expansión mejora el valor de la función objetivo, entonces $\mathbf{x}_e$ reemplaza a $\mathbf{x}_D$. Este nuevo conjunto de $D+1$ puntos es usado para repetir el procedimiento.
  - En el caso de que $\mathbf{x}_e$ no mejore el valor de $\mathbf{x_0}$, entonces $\mathbf{x}_D$ es reemplazado por $\mathbf{x}_r$ (obsérvese que $\mathbf{x}_D$ es el peor valor de la función objetivo de los $D+1$ puntos después de la ordenación)
  - Si $\mathbf{x}_r$ no mejora al punto $\mathbf{x_0}$ en primer lugar, $\mathbf{x}_r$, entonces $\mathbf{x}_r$ es comparado con el siguiente peor punto,$\mathbf{x}_{D-\mathbf{1}}$.
  - Si $\mathbf{x}_r$ es mejor que $\mathbf{x}_{D-\mathbf{1}}$, entonces $\mathbf{x}_r$ se reemplaza $\mathbf{x}_D$.
  - Si, sin pese a todo, $\mathbf{x}_r$ es peor que $\mathbf{x}_{D-\mathbf{1}}$, se usa un tercer factor de escalado, F_3, para encoger el simplex entero.

Nelder-Mead Method

```
while(not converged) {
  sort(x_i, D + 1);
  x_m = 0
  for (i = 0; i < D; i++) {
    x_m = x_m + x_i;
  }
  x_m = x_m / D;
  x_r = x_m + F_1(x_m - x_0)
  if (f(x_r) < f(x_0)) {
    x_e = x_m + F_2(x_m - x_0);
    if (f(x_e) < f(x_0))
      x_D = x_e;
    else
      x_D = x_r;
  }
  else if (f(x_r) < f(x_{D-1})) {
    x_D = x_r;
  }
  else {
    if (f(x_r) < f(x_D)) {
    x_D = x_r;
    x_c = x_m + F_3(x_m - x_D);
    }
    else {
    x_c = x_m - F_3(x_m - x_D);
    }
    if (f(x_c) < f(x_D)) {
    x_D = x_c;
    }
    else {
      for (i = 1; i <= D; i++) {
        x_i = 0,5 * (x_0 + x_i);
      }
    }
}
```

Nelder-Mead

- En la figura se muestra los mecanismos básicos de reflexión y expansión del método de Nelder-Mead para una función objetivo en un espacio de $2D$ y en la fig 1 de la transparencia siguiente se muestra un secuencia de puntos de búsqueda .

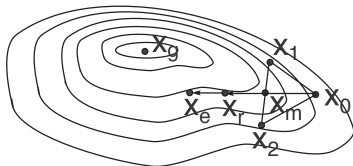
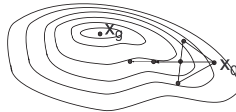
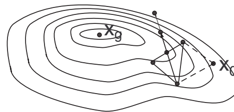


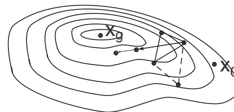
Figura: Nelder-Mead: mecanismo de reflexión y expansión para $D = 2$.

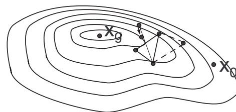
Nelder-Mead

Nelder-Mead secuencia de búsqueda.









Nelder-Mead

- El método de Nelder-Mead fué uno de los primeros métodos de optimización que han usado de forma extensiva el vector diferencia para explorar el landscape de una función de coste.
- Una de las ventajas de este método es la posibilidad de expandir y contraer el simplex para adaptar la búsqueda a la superficie local actual de la función objetivo. Esto permite que el algoritmo adapte el tamaño de su paso de avance a las condiciones locales de la función.
- El método de Nelder-Mead restringe el número de puntos de muestreo a $D + 1$ puntos (D es la dimensión del espacio de búsqueda).
- Esto limita sus ventajas cuando la funcin objetivo es compleja.

Controlled Random Search (CRS)

- El método de la búsqueda aleatoria controlada (Controlled Random Search, CRS) fué propuesto por Price en 1978.
- Este método también usa el vector de diferencias para operaciones de reflexión.
- Emplea el simplex de Nelder-Mead de $D + 1$ puntos obtenidos por muestreo aleatorio de una población de $N_p > D + 1$ vectores tal y como se puede ver en la figura siguiente.

Controlled Random Search (CRS)

- La reflexión a través del centroide genera un nuevo punto a explorar $\mathbf{x}_r$.
- Si este punto es mejor que el peor de los puntos de la población actual $\mathbf{x}_w$, entonces $\mathbf{x}_r$ reemplaza a $\mathbf{x}_w$ en el conjunto de la población

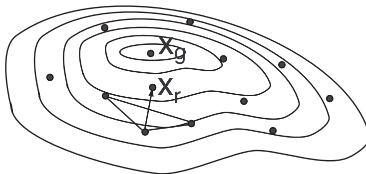


Figura: El método CRS usa la reflexión de Nelder-Mead a partir de una población de puntos.

Controlled Random Search (CRS)

Nelder-Mead Method

while(not converged) {
  sort(x_i, N_p);
  j=**rand**(N_p);
  $\mathbf{x}_m = \mathbf{x}_j$;
  for ($i = 0; i < D; i++$) {
    j=**rand**(N_p);
    $\mathbf{x}_m = \mathbf{x}_m + \mathbf{x}_j$;
  }
  $\mathbf{x}_m = \mathbf{x}_m / D$;
  $\mathbf{x}_r = \mathbf{x}_m + F_1(\mathbf{x}_m - \mathbf{x}_j)$;
  if (**bounds-ok**($\mathbf{x}_r == TRUE$))
  {
    if ($f(\mathbf{x}_r) < f(\mathbf{x}_D)$)
    {
      $\mathbf{x}_D = \mathbf{x}_r$;
    }
  }
}

Controlled Random Search (CRS)

- En el método CRS la generación del vector de diferencias través de la reflexión puede considerarse como una forma aritmética de obtener la recombinación.
- El reemplazo continuo del peor punto posible en el conjunto de la población introduce un mecanismo de presión selectiva que tiende a forzar al método a una rápida convergencia.
- Esta convergencia rápida tiene el inconveniente de que puede ser en ocasiones prematura.
- Se han propuesto diversas mejoras sobre el algoritmo CRS básico (Ali et al. (1997) y Ali y Törn (2004)).

Algoritmos evolutivos

- Estos métodos utilizan en la búsqueda de soluciones mejores operaciones del tipo:
  - recombinación,
  - mutación
  - supervivencia o selección del mejor.
- Debido a que imitan a la evolución Darwinista en la naturaleza se les denomina *algoritmos evolutivos (evolutionary algorithms)* (EAs).
- Aunque se les agrupa en una misma clase de métodos existen diferencias muy significativas entre ellos,

Estrategias de evolución (Evolution strategies, ESs)

- Las ES muestrean la función objetivo en muchos puntos diferentes, pero a diferencia de los enfoques multi-start donde cada punto base evoluciona aisladamente, los puntos de una población en las estrategias evolutivas se influyen uno a otro mediante la recombinación.
- Se comienza con una población de μ vectores progenitores (puntos), el algoritmo ES genera una población de vectores descendientes $\lambda \geq \mu$ mediante la recombinación entre vectores progenitores escogidos aleatoriamente.

Estrategias de evolución (Evolution strategies, ESs)

- La recombinación puede ser discreta (algunos parámetros son seleccionados de uno de los progenitores y los otros se toman del otro progenitor) o intermedia (promediando los parámetros de ambos progenitores) (Bäck 1996).
- Después de la recombinación, cada uno de los puntos descendientes es mutado mediante la adición de una desviación o perturbación aleatoria $\Delta\mathbf{x}$, típicamente una desviación Gaussiana aleatoria con media cero y covarianza conocida.
- Después de la mutación, todos los λ puntos descendientes son evaluados, y de los puntos descendientes la estrategia evolutiva $(\mu, \lambda) - ES$ selecciona los μ mejores descendientes que serán los que pasarán a ser progenitores en la próxima generación.
- Otros enfoques alternativos selecionan los μ mejores vectores de las poblaciones combinadas de progenitores y descendientes, se denomina estrategia $(\mu + \lambda) - ES$.

Estrategias de evolución (Evolution strategies, ESs)

Evolution strategy Method

Initialization();
while (not converged)
{
  for $(i = 0; i < \lambda; i++)$
  {
    $p_1(i) =$ **rand**(μ);
    $p_2(i) =$ **rand**(μ);
    $c_1(i) =$ **recombine**$(p_1(i), p_2(i))$;
    $c_1(i) =$ **mutate**$(c_1(i))$
  {
  selection();
}

Estrategias de evolución (Evolution strategies, ESs)

- Las ESs constituyen uno de los mejores optimizadores globales, pero siguen sin resolver el problema de la adaptación del tamaño del paso de avance, al menos en las implementaciones más simples.
- Si se permite que la distribución Gaussiana simétrica se convierta en una distribución Gaussiana elipsoidal, el ES puede asignar diferentes tamaños de paso en cada dimensión.
- Además, la matriz de covarianza permite a la distr. Gaussiana elipsoidal de mutación rotar para adaptarse mejor a la topografía de las funciones objetivo non-descomponibles.

Estrategias de evolución (Evolution strategies, ESs)

- Un función descomponible siempre puede reescribirse en la forma (Salomon 1996)

$$f(\mathbf{x}) = \sum_{i=0}^{D-1} f_i(x_i) \qquad (15)$$

- Debido a que las funciones descomponibles no tienen términos cruzados, sus parámetros pueden ser optimizados de independientemente.
- Por tanto, la descomponibilidad reemplaza la tarea de optimizar una función con D dimensiones en el problema mucho más simple de optimizar D funciones unidimensionales.

- El hiper-elipsoide es un caso simple de función descomponible:

$$f(\mathbf{x}) = \sum_{i=0}^{D-1} \alpha_i x_i \tag{16}$$

- Si se rota el hiper-elipsoide en todas las dimensiones, se hace imposible el optimizar un sólo parámetro independientemente de los otros.
- Esta dependencia entre parámetros, se denomina a menudo como epistasis, una expresión tomada de la biología. Salomon (1996) demostró que a menos que un optimizador enfoque el hecho de la dependencia de parámetros sus prestaciones en funciones objetivo epistáticas se degradarán sustancialmente.

Algoritmos Genéticos (Genetic Algorithms, GAs)

- Los más conocidos y usados de los métodos evolutivos son los algoritmos genéticos (GAs) propuestos por Goldberg en 1989.
- En estos métodos el conjunto de puntos que forma la población S evoluciona de una generación a otra mediante el reemplazo de subconjuntos de S sucesivamente.
- En los algoritmos genéticos, toda solución $\mathbf{x}$ es codificada por lo general usando una representación mediante una cadena binaria que se asemeja a la forma en la que los cromosomas codifican la información genética.
- Es posible también utilizar una versión de los algoritmos genéticos denominada real coded GA que representa variables continuas. Los tres conceptos básicos implicados en la versión básica de un algoritmo genético son:
    - Evaluación.
    - Selección estocástica
    - Reproducción

Algoritmos Genéticos (Genetic Algorithms, GAs)

Un vistazo rápido

- Pioneros: J. Holland, K. DeJong, D. Goldberg
- Se han aplicado típicamente en:
    - Optimización discreta (combinatoria)
- Entre sus características están:
    - No es demasiado rápida.
    - Buen heurístico para problemas combinatorios.
    - Enfatiza la combinación de información entre buenos progenitores (crossover)
    - Existen muchas variantes, ej., diferentes modelos de reproducción, de operadores, etc.

Algoritmo Genético Simple o Canónico (SGAs, CGAs)

- Se denomina así a la versión original del Algoritmo Genético que propuso Holland y que se conoce como Algoritmo Genético Simple (simple genetic algorithm, SGA).
- Los algoritmos genéticos codifican las soluciones potenciales en los cromosomas y les aplican operadores genéticos a los mismos.
- Esto equivale a transformar el problema original de un espacio a otro.
- La representación genética es importante en el éxito del algoritmo genético, ya que puede hacerlo más fácil o más difícil de resolver.
- Las poblaciones que maneja consisten en soluciones codificadas de problemas.
- La búsqueda de buenas soluciones es realizada en el espacio de soluciones codificadas.
- La manipulación de poblaciones usa los operadores de: selección, cruce y mutación.

Algoritmo Genético Simple o Canónico (SGAs, CGAs)

Características importantes:

- No trabajan con los objetos, sino con una codificación de los mismos.
- Los GA realizan una búsqueda mediante toda una generación de objetos, no buscan un único elemento.
- Utilizan una función de salud que nos da información de lo adaptados que están.
- Las reglas de transición son probabilísticas no deterministas.

Algoritmo Genético Simple o Canónico

1. Generar la población inicicial $P(0)$ aleatoriamente y poner $i = 0$;
2. **Repetir**
   - Evaluar la salud de cada individuo (fitness) en $P(i)$
   - *Seleccionar* los padres de $P(i)$ en base a su salud del siguiente modo: Dados los valores de salud de n individuos $f_1, f_2, \ldots, f_n$. Seleccionar un individuo i con probabilidad

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{17}$$

     Este método se denomina selección de la ruleta y es un método de selección proporcional al fitness.
   - Se aplica el *operador cruce* a los padres seleccionados.
   - Se aplica el *operador de mutación* a los nuevos individuos credos después del cruce.
   - Se reemplazan los padres por los descendientes para producir la generación $P(i + 1)$.

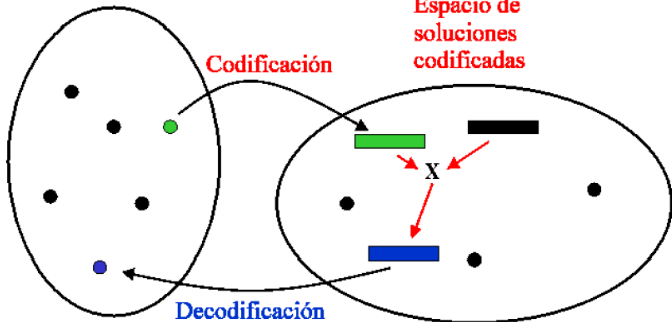
hasta que la condición de parada se verifique.

Algoritmo Genético Simple (SGAs)

Conceptos involucrados:

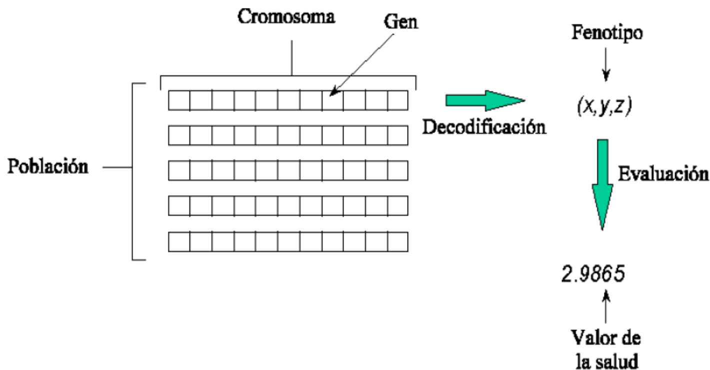
- Representación de la información: mediante cadenas binarias (Binary strings)
- Recombinación: mediante de N-puntos o uniforme.
- Mutación: de bit (bit-flipping) con probabilidad prefijada
- Selección de progenitores: proporcional al valor de salud (fitness)
- Selección de supervivientes: todos los descendientes reemplazan a sus progenitores
- Especialidad: enfatiza el cruce (crossover)

Algoritmo Genético Simple (SGAs)



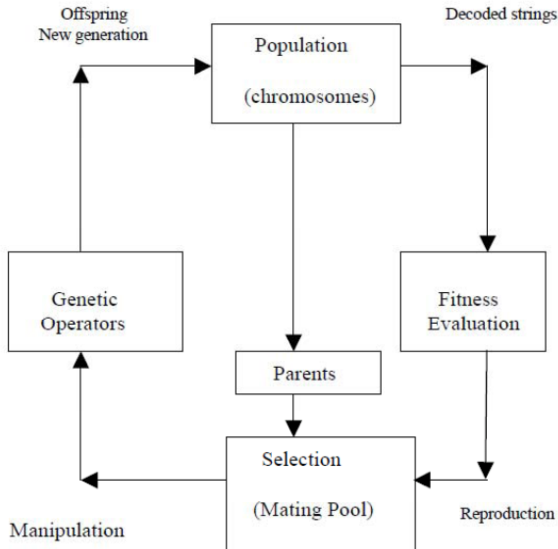
Algoritmo Genético Simple (SGAs)

Términos usuales utilizados en los algoritmos genéticos.



Algoritmo Genético Simple

Ciclo básico



La idea básica de cualquier operador de recombinación (cruce) es la herencia de información (genes) desde dos o más progenitores a sus descendientes. Aunque la mayoría de los operadores de recombinación usan dos progenitores, pueden usarse múltiples padres. Por lo general se producen dos descendientes en el operador de recombinación pero pueden ser múltiples.

- Recombinación de *Real-Valued Vectors*.
  Este tipo de operador se usa en el caso de tener vectores de números reales. Se usa mucho en algoritmos ES y RCGA.
- Recombinación de *Binary Strings*.
  El operador de recombinación más común para arrays binarios (binary strings) incluye el cruce de k puntos ($k > 1$) y el cruce uniforme, si bien existen otras variantes.

Recombinación de *Real-Valued Vectors*

- Recombinación discreta.
  - Un vector descendiente tiene componentes que provienen de dos o más vectores progenitores. No hay cambios en ningún componente elemental. Por ejemplo, dados dos padres $\mathbf{x} = (x_1, \ldots, x_n)$ e $\mathbf{y} = (y_1, \ldots, y_n)$. Los descendientes $\mathbf{x}' = (x'_1, \ldots, x'_n)$ e $\mathbf{y}' = (y'_1, \ldots, y'_n)$ pueden obtenerse como:

$$x'_i = \begin{cases} x_i & \text{con probabilidad de recombinación } p_r \\ y_i & \text{en otro caso} \end{cases} \tag{18}$$

    y donde $\mathbf{y}'$ es el complementario de $\mathbf{x}'$.
- Recombinación intermedia.
  - Cada componente del vector descendiente se obtiene como combinación lineal (promedio) de los componentes correspondientes de los progenitores. Por ejemplo, dados dos padres $\mathbf{x}$ e $\mathbf{y}$, los descendientes $\mathbf{x}'$ e $\mathbf{y}'$ pueden obtenerse como:

$$x'_i = x_i + \alpha(y_i - x_i) \tag{19}$$

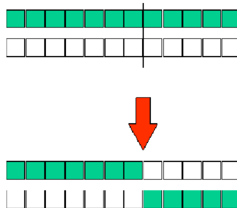
    donde α es un parámetro de ponderación en el intervalo (0,1), tradicionalmente se toma 0,5, pero puede ser generado aleatoriamente.

Recombinación de *Binary Strings*

GAs

- Cruce multi-punto.
  - Este cruce puede aplicarse a strings de cualquier alfabeto. Dados dos padres de longitud n, se generan k números aleatorios $r_1, r_2, \ldots, r_k$ según una distribución uniforme (sin repetición). Se originan entonces unos descendientes tomando segmentos (separados) de los padres alternativamente, ej., el primer segmento del primer padre, el segundo del segundo padre, el tercero del primer padre y así sucesivamente.
    - Por ejemplo, para un cruce de 3 puntos en $1, 4, 6$ de dos padres 00000000 y 11111111 producirá dos descendientes 01110011 y 10001100.
- Cruce uniforme.
  - Este cruce es aplicable también a strings de cualquier alfabeto. Se generan los descendientes tomando cada bit o carácter de uno de sus padres correspondientes. El padre del que se toma el bit o carácter se escoge aleatoriamente según una distribución de probabilidad uniforme.

- Operador de cruce para cadenas binarias mono-punto
  - Se escogen dos individuos de la población mediante el operador de selección.
  - Se escoge aleatoriamente un lugar o punto de cruce.
  - Los valores de las dos cadenas se intercambian en este punto.
  - Recombinando porciones de buenos individuos se van creando individuos aún mejores .



Los operadores de mutación pueden ser:

- Operadores de mutación para *Real-Valued Vectors*.
  Los operadores de mutación para vectores de valores reales suelen basarse en ciertas distribuciones de probabilidad, tales como la distribución uniforme, la lognormal, la normal o Gaussiana y la de Cauchy.
- Operadores de mutación para *Binary Strings*.
  Los operadores de mutación para cadenas binarias suelen ser mucho más simples. Usualmente es una mera operación de cambio de bit (bit-fliping operation).

Mutación de *Real-Valued Vectors*

- Mutación Gaussiana.
  - Los descendientes se generan añadiendo un número aleatorio distribuido según una Gaussiana con media 0 y una desviación estándar σ a los progenitores. Por ejemplo, dado un padre $\mathbf{x} = (x_1, \ldots, x_n)$ el descendiente se produce de la siguiente forma

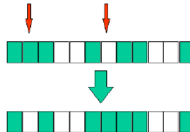
$$x_i' = x_i + N_i(0, \sigma_i) \tag{20}$$

    donde $N_i(0, \sigma_i)$ es un número aleatorio distribuido según una normal con media 0 y desviación estándar σ_i. Los n números aleatorios son generados de forma independiente para cada una de las dimensiones.
- Mutación Cauchy.
  - La Mutación Cauchy difiere de la gaussiana en la distribución de probabilidad usada para generar los números aleatorios.

Mutación de *Binary Strings*

- Mutación binaria (bit-flipping).
  - La mutación binaria (Bit-flipping mutation) consiste simplemente en cambiar el bit de 0 to 1 o de 1 a 0 con una cierta probabilidad. Esta probabilidad a menudo se denomina *probabilidad de mutación* o *tasa de mutación*.
  - La mutación binaria puede ser generalizada a strings o cadenas de cualquier alfabeto. Esta mutación generalizada opera del siguiente modo: para cada carácter (alelo) en una cadena, se reemplaza con cualquier otro caracter aleatoriamente elegido del alfabeto (distinto del que se va a reemplazar) con una cierta probabilidad de mutación.
- Mutación aleatoria.
  - La mutación aleatoria (Random Bit) no conmuta un bit. Reemplaza un bit por 0 o 1 con igual probabilidad (es decir, 0,5 respectivamente). En su versión generalizada cada carácter (alelo) en una cadena, se reemplaza con un carácter aleatoriamente elegido (puede ser el mismo que va a ser reemplazado) del alfabeto con una cierta probabilidad de mutación.

- Para una cadena binaria
  - Con una cierta probabilidad, muy baja, una cierta porción de los nuevos individuos pueden mutar sus bits.
  - Su propósito es mantener la diversidad dentro de la población y prevenir la convergencia prematura.



El esquema de selección determina la probabilidad de que un individuo sea seleccionado como progenitor para producir descendientes mediante los operadores de recombinación y/o mutación.
Para buscar individuos progresivamente mejores, los individuos más adaptados deberían tener mayores probabilidades de ser seleccionados (y viceversa).
Los tres métodos más usados en los esquemas de selección son:

- *Método de la ruleta* (roulette wheel selection) también denominado selección proporcional al fitness.
- *Selección basada en el rango* (rank-based selection).
- *Selección por torneo* (tournament selection).

Selección por el método de la ruleta

GAs

- Método de la ruleta
  - Dados los valores de salud de n individuos $f_1, f_2, \ldots, f_n$. La probabilidad de seleccionar un individuo i se calcula según

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{21}$$

  - Es decir se calcula la probabilidad de selección directamente a partir de los valores de fitness de los elementos de la población.
  - Este método puede originar problemas en ciertos casos. Por ejemplo, cuando en la población inicial algunos valores son significativamente mejores que los demás (pero no necesariamente los mejores) lo que hace que predominen con rapidez pero no exploran otras soluciones potenciales.

- Selección basada en rango
  - La selección basada en rango no calcula la probabilidad de selección directamente a partir de los fitness. Ordena a los individuos en primer lugar de acuerdo a su valor de fitness y después selecciona de acuerdo a su rango en vez de los valores de fitness.
  - Así se puede mantener una presión selectiva constante y evitar los problemas del método de la ruleta.
  - Hay diferentes esquemas basados en el rango, dos de ellos son:
    - Suponer que el mejor individuo de la población se ordena en la primera posición. La probabilidad de selección puede calcularse linealmente del siguiente modo (Baker, 1985):

$$p_i = \frac{1}{n}\left(\nu_{max} - (\nu_{max} - \nu_{min})\frac{i-1}{n-1}\right) \qquad (22)$$

      donde n es el tamaño de la población, ν_{max} y ν_{min} son dos parámetros tal que $\nu_{max} \geqslant \nu_{min} \geqslant 0$ y $\nu_{max} + \nu_{min} = 2$
    - Un esquema de presión selectiva no lineal (Yao, 1993):

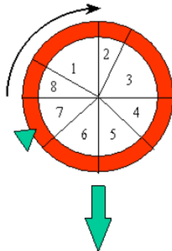
$$p_i = \frac{i}{\sum_{j=1}^{n} j} \qquad (23)$$

- Selección por torneo
  - Las selección de ruleta y basada en rango se basan en la información global de toda la población lo que incrementa las comunicaciones en el caso de que se desee paralelizar el algoritmo.
  - El sistema de torneo sólo requiere conocer parte de la población para calcular la probabilidad de selección de un individuo, por lo que diferentes individuos pueden calcular la suya en paralelo.
  - En el esquema de torneo de tamaño 3, primero se selecciona un individuo i_1 aleatoriamente. Entonces se selecciona i_2 también aleatoriamente pero debe diferir del i_1 por una cantidad de fitness θ. De forma aleatoria se selecciona i_3 que debe también diferir de i_1 y la mitad de las veces diferir de i_2 también, todos por θ.
  - i_2 compite con i_3 primero. Posteriormente el ganador compite con i_1. El ganador es identificado por la probabilidad de aceptación de Boltzmann. La probabilidad de que un individuo x gane sobre y es:

$$P(x, y) = \frac{1}{1 - e^{(f_x - f_y)/T}} \tag{24}$$

donde T es la temperatura. (Se busca maximizar el fitness)

Método de la ruleta.



Ejemplo

Función de fitness simple

Función a optimizar: $f(x) = x^2$.

Individuo Codificado	Valor(Fenotipo)	Salud asociada
01101	13	169
10100	20	400
00100	4	16
00111	7	49

Tab. 2.1: *Fenotipos y salud asociada.*

Individuo	Probabilidad	N° Copias Esperado
01101	169/634= 0,27	1,1
10100	400/634= 0,63	2,5
00100	16/634= 0,02	0,1
00111	49/634= 0,08	0,3

Tab. 2.2: *Número de copias esperado.*

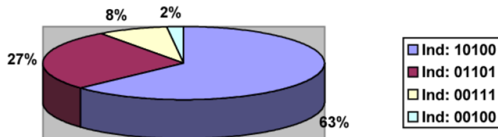
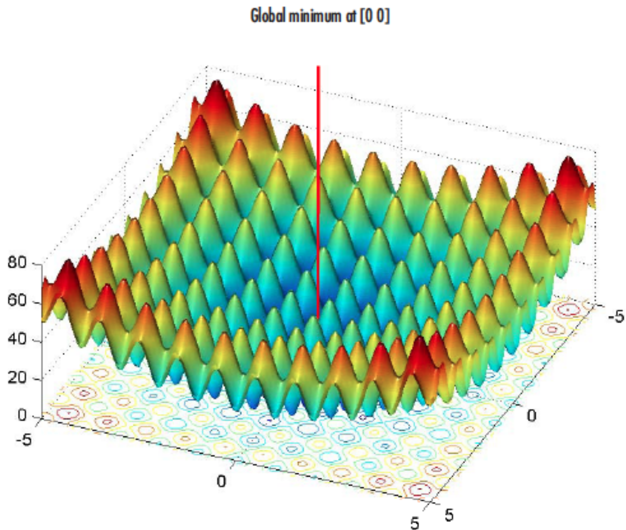


Fig. 2.7: *Función de salud.*

Ejemplo de función difícil

- Rastrigin.

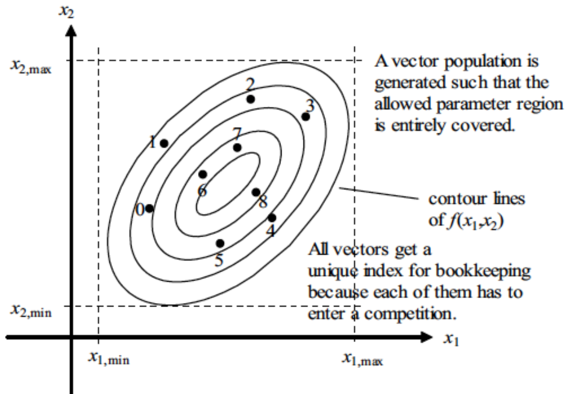


Idea intuitiva del método:

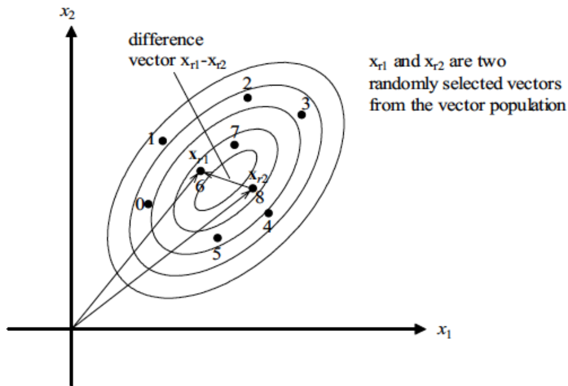
- Generación de una población inicial
- Repetir
  - Generar una nueva población mediante perturbaciones (una especie de mutación).
  - Cruzar la población perturbada .
  - Realizar la selección de los elementos que pasan a la nueva población.

  hasta que se verifica la condición de parada

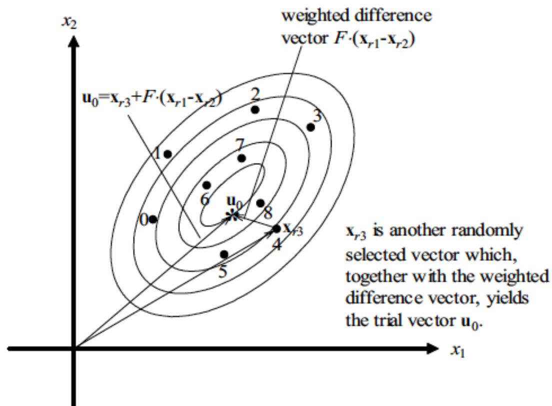
- Se genera en primer lugar una población inicial (inicialización).



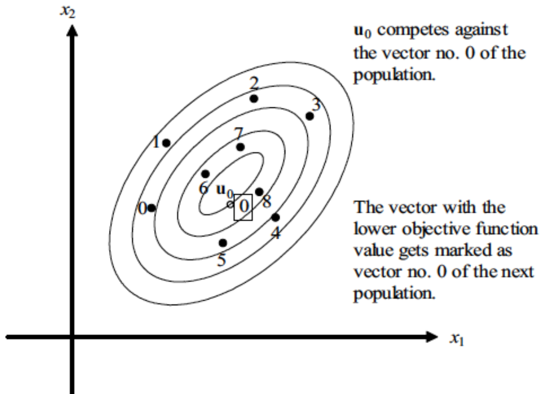
- Se genera la perturbación $\mathbf{x}_{r_1} - \mathbf{x}_{r_2}$, a partir de dos vectores de la población que se eligen aleatoriamente (pero distintos).



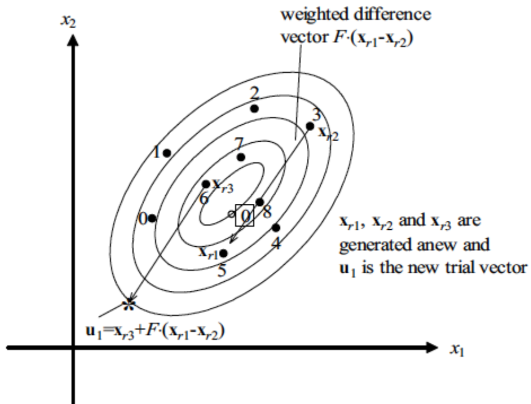
- La perturbación se escala por un factor F y se suma aun tercer vector x_{r_3} escogido aleatoriamente (distinto de x_{r_1} y x_{r_2}, con lo que se crea un nuevo vector pertubado o por similitud una mutación .



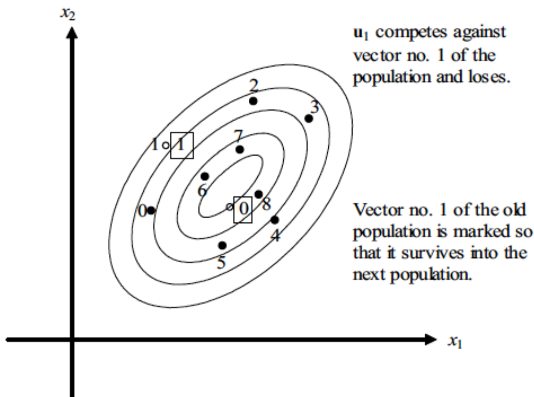
- Se realiza la selección.



- Se genera un segundo vector perturbado .



- Se realiza la selección, que en este caso no resulta favorable manteniéndose el elemento anterior de la población y descartándose el nuevo.



DE Method

```
Initialization();
while (convergence criterion not yet met) {
   // x_i defines a vector of the current vector population
   // y_i defines a vector of the new vector population
   for (i = 0; i < NP; i++) {
      r_1 = rand(NP); //select a random index from 1, 2, ..., N_p
      r_2 = rand(NP); //select a random index from 1, 2, ..., N_p
      r_3 = rand(NP); //select a random index from 1, 2, ..., N_p
      u_i = x_r3 + F * (x_r1 - x_r2);
      if (f(u_i) <= f(x_i))
         y_i = u_i;
      else
         y_i = x_i;
   }
} //end while
```

- En DE se suelen mantener un par de poblaciones de vectores, las cuales tienen N_p vectores D-dimensionales de parámetros de números reales.
  - La población actual, P_x, está compuesta por aquellos vectores, $x_{i,g}$, que han sido encontrados aceptables en la inicialización o en la comparación con otro vector en una iteración anterior.

$$\begin{aligned} P_{x,g} &= \{x_{i,g}\}, \; i = 0, 1, \ldots, N_p - 1, \; g = 0, 1, \ldots, g_{max}, \quad (25) \\ x_{i,g} &= (x_{j,i,g}), \; j = 0, 1, \ldots, D - 1. \end{aligned}$$

  - Los índices se han comenzado por 0 para simplificar su trabajo con arrays y la arimética de módulos. El índice, $g = 0, 1, \ldots, g_{max}$, indica la generación a la que pertenece el vector. A cada vector se le asigna un índice de población, i, que va desde 0 a $N_p - 1$. Los parámetros en los vectores están indexados con j, que va desde 0 a $D - 1$.

- Cont.
  - Una vez inicializado, DE muta mediante los vectores elegidos aleatoriamente para producir una población intermedia, $P_{v,g}$, de N_p vectores perturbados o mutados, $v_{i,g}$:

$$\begin{aligned} P_{v,g} &= \{v_{i,g}\},\ i = 0, 1, \ldots, N_p - 1,\ g = 0, 1, \ldots, g_{max}, \\ v_{i,g} &= (v_{j,i,g}),\ j = 0, 1, \ldots, D - 1. \end{aligned} \quad (26)$$

  - Cada vector en la población actual se recombina posteriormente con un vector mutado para producir la población nueva a probar, P_u, de N_p vectores de prueba, $u_{i,g}$:

$$\begin{aligned} P_{u,g} &= \{u_{i,g}\},\ i = 0, 1, \ldots, N_p - 1,\ g = 0, 1, \ldots, g_{max}, \\ u_{i,g} &= (u_{j,i,g}),\ j = 0, 1, \ldots, D - 1. \end{aligned} \quad (27)$$

  - Durante la recombinación, los vectores de prueba sobreescriben la población mutante para que un único array pueda contener las dos poblaciones.

- Antes de que se inicialice la población, deben especificarse los límites superior e inferior de cada parámetro.
  - Estos $2D$ valores se guardan en dos vectores, D-dimensionales, de inicialización, b_L y b_U, donde los subídices L y U indican los límites inferior y superior respectivamente.
  - Una vez definidos los límites, un generador de números aleatorios asigna a cada parámetro de cada vector un valor situado en el rango preestablecido. Por ejemplo, el valor inicial ($g = 0$) del j-ésimo parámetro del vector i-ésimo es

$$x_{j,i,0} = rand_j(0,1).(b_{j,U} - b_{j,L}) + b_{j,L} \tag{28}$$

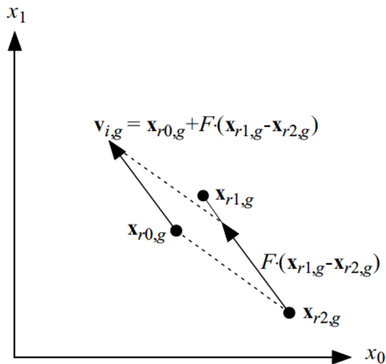
  - El generador de números aleatorios, $rand_j(0,1)$, devuelve un número aleatorio distribuido según una distribución uniform en el rango $[0,1)$, es decir, $0 \leq rand_j(0,1) < 1$.
  - Incluso si la variable es discreta, debe ser inicializada con un valor real dado que el método DE internamente trata a todas las variables como valores en coma flotante independientemente de su tipo.

- Una vez inicializado, DE muta y recombina la población para producir un población de N_p vectores de prueba N_p.
  - Se genera una mutación diferencial mediante el vector de diferencia (aleatoriamente obtenido) que escalado se añade a un tercer vector para crear el vector mutado o perturbado, $v_{i,g}$:

$$v_{i,g} = x_{r_0,g} + F(x_{r_1,g} - x_{r_2,g}) \tag{29}$$

  - El factor de escala , $F \in (0, 1+)$, es un número real positivo que controla la tasa a la que la población. Si bien no existe un límite superior para F, los valores efectivos no suelen ser mayores que 1,0.
  - El índice del vector base, r_0, puede determinarse de diferentes formas, pero puede suponerse que es aleatoriamente elegido y que es diferente del iíndice del vector objetivo, i. Los índices usados en el vector de diferencia son distintos entre sí, del vector base y del vector objetivo, y son elegidos aleatoriamente, r_1 y r_2, para cada vector que se muta.

- En la figura se muestra la construcción del vector mutante, $v_{i,g}$, en un espacio de dos dimensiones



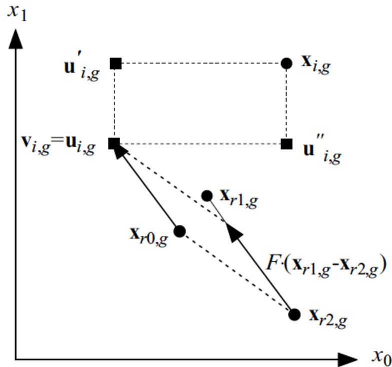
- Para complementar la estrategia de búsqueda de la mutación diferencial, DE utiliza además una estrategia de cruce uniforme.
  - La recombinación discreta, construye los vectores de prueba a partir de los parámetros tomados de dos vectores diferentes. En el caso particular de DE se cruzan los vectores de la población inicial con los vectores mutantes :

$$u_{i,g} = u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } (rand_j(0,1) \leq Cr) \text{ or } (j < j_{rand}) \\ x_{j,i,g} & \text{otherwise} \end{cases} \quad (30)$$

  - La probabilidad de cruce, $Cr \in [0,1]$, es un valor definido por el usuario que controla la fracción de valores de los parámetros que son copiados desde el vector mutante.
  - Para determinar que fuente contribuye a un parámetro dado, el cruce uniforme compara Cr con la salida de un generador de números aleatorios, $rand_j(0,1)$.

- Cont.
  - Si el número aleatorio es menor o igual que Cr, el parámetro a probar se hereda del mutante, $v_{i,g}$; de otro modo, el parámetro es copiado desde el vector, $x_{i,g}$.
  - Además, el parámetro de prueba con un índice elegido aleatoriamente , j_{rand}, es tomado del mutante para asegurar que el vector vector de prueba no duplica a $x_{i,g}$.
  - Debido a esto, Cr sólo aproxima la probabilidad verdadera, p_{Cr}, de que un parámetro de prueba será heredado del mutante.

- En la figura se muestra los posibles vectores de prueba que pueden resultar a partir del cruce uniforme del vector mutante, $v_{i,g}$, con el vector xi, g.



- Estrategia de selección.
  - Si el vector de prueba, $u_{i,g}$, tiene un valor de la función objetivo igual o menor que el valor de la función para el vector objetivo, $x_{i,g}$, se reemplaza el vector objetivo en la próxima generación;
  - En otro caso, el objetivo retiene su lugar en la población por al menos una generación más.
  - Al comparar cada vector de prueba con el vector objetivo del cual hereda sus parámetros, DE integra de forma mas estrecha las operaciones de selección y recombinación que otros algoritmos evolutivos:

$$x_{i,g+1} = \begin{cases} u_{i,g} & \text{if } \left(f(u_{i,g}) \leq f(x_{i,g})\right) \\ x_{i,g} & \text{otherwise} \end{cases} \tag{31}$$

  - Una vez que la nueva población sustituye a la previa, los procesos de mutación, recombinación y selección se repite hasta que se localiza el óptimo, o se satisface un criterio de terminación preespecificado, por ejemplo que se alcanza un cierto número de generaciones máximo, g_{max}.

Differential Evolution

Differential Evolution Method

while(termination criteria not met) {
  // generate a trial population
  for $(i = 0; i < N_p; i++)$ {
    do $r_0 = floor(rand(0,1) * N_p)$; **while** $(r_0 == i)$;
    do $r_1 = floor(rand(0,1) * N_p)$; **while** $(r_1 == r_0$ **or** $r_1 == i)$;
    do $r_2 = floor(rand(0,1) * N_p)$; **while** $(r_2 == r_1$ **or** $r_2 == r_0$ **or** $r_2 == i)$;
    $j_rand = floor(D * rand(0,1))$;
    // generate a trial vector
    for $(j = 0; j < D; j++)$ {
      if $(rand(0,1) <= Cr$ **or** $j == j_rand)$ {
        $\mathbf{u}_{j,i} = \mathbf{x}_{j,r_0} + F * (\mathbf{x}_{j,r1} - \mathbf{x}_{j,r2})$;
        //check for out-of-bounds ? }
      else { $\mathbf{u}_{j,i} = \mathbf{x}j, i$; }
    }
  }

⋮

Differential Evolution Method (cont)

⋮

// select the next generation
for $(i = 0; i < Np; i++)$ {
  if $(f(\mathbf{u}_i) <= f(\mathbf{x}_i))\mathbf{x}_i = \mathbf{u}_i;$ }
}

Particle Swarm Optimization (PSO)

Concepto

- Idea básica:
  - El método conocido como Particle Swarm Optimization (PSO), fue propuesto por Eberhart y Kennedy en 1995, es una forma de imitar la inteligencia grupal (swarm intelligence) en la cual se simula el comportamiento de un sistema social biológico como puede ser una bandada de pájaros.
  - Cuando un grupo busca comida, sus individuos se extienden por el entorno y se mueven por él, independientemente.
  - Cada individuo tiene un cierto grado de libertad o aleatoriedad en sus movimientos lo cual les permite encontrar acumulaciones de comida.
  - De esta forma, antes o depués, alguno de sus miembros encontrará algo comestible y, siendo sociables, anunciarán a sus vecinos que se aproximarán también a la fuente de alimento.

Particle Swarm Optimization (PSO)

Cooperacion versus competicion

- En general los algoritmos genéticos y otros de la familia de los algoritmos evolutivos presentan una característica notablemente diferente de los PSO al menos en la versión clásica de estos: y es que en los PSO no se utiliza selección.
- La idea subyacente es que las partículas con menos éxito en un momento dado pueden ser las que más éxito tengan en iteraciones futuras. Por lo que las partículas con peores prestaciones se preservan. Bastantes experimentos justifican esta decisión.
- No obstante se han desarrollado versiones del PSO que utilizan mecanismos de selección.

Particle Swarm Optimization (PSO)

Propagación de un rumor

- Es bastante intuitivo, y tradicional, el modelar una red de transmisión de información entre individuos como un grafo, a veces denominado grafo de influencia. Cada nodo del grafo representa un individuo, y un enlace entre nodos A y B representa el enlace o flujo de información entre los dos individuos, es decir A informa a B.
- Estos enlaces pueden no ser constantes. En particular en los enjambres o bandadas (swarms) pueden cambiar en cada momento (imaginemos que se informa a los tres más cercanos a uno dado).
- Se ha estudiado el fenómeno de avalanchas de información y su influencia en fenómenos de aprendizaje.
- No obstante un modelo muy simple es suponer que con cada incremento de tiempo (ciclo) cada partícula elige aleatoriamente un cierto número de otras partículas a las que informar. Se puede calcular el valor mínimo de este número para tener una alta probabilidad de que cualquier información es recibida al menos una vez cada cierto tiempo.

Particle Swarm Optimization (PSO)

Concepto de sitio o lugar interesante

- En PSO, un sitio interesante corresponde al menos a un óptimo local de una cierta función definida en el espacio de estados.
- En PSO el punto de partida de la conducta cooperativa es que: cada partícula es capaz de comunicar a algunas otras la posición y calidad de la mejor solución (pose, site) que conoce, la calidad se interpreta como el valor de la función.
- Denominemos descendientes a las partículas conectadas a una dada en sentido descendente de los enlaces de informaciónel grupo de receptores. En un cierto momento una partícula puede pertenecer a varios grupos receptores y tener varios informantes que le informan de varios lugares más o menos buenos.
- Esta información se puede usar para definir el próximo desplazamiento de la partícula. En la naturaleza la síntesis de estas informaciones no esta claro como se realiza pero es fácil componerlas de forma lineal. Y además ha demostrado funcionar de fiorma muy efectiva. en problemas de optimización.

Particle Swarm Optimization (PSO)

- Conceptos de velocidad y posición.
  - En el método Particle Swarm Optimization, se simula un grupo de partículas (individuos) en un espacio de búsqueda n-dimensional, donde cada partícula i tiene una posición $\mathbf{x}_i \in \mathcal{R}^n$ y una velocidad $\mathbf{v}_i \in \mathcal{R}^n$
  - La posición $\mathbf{x}_i$ se corresponde con los genotipos, y, en la mayoría de los casos, también con las soluciones potenciales al problema.
  - Sin embargo esto no es estrictamente necesario, y de forma general, se podría introducir cualquier forma de mapeo de genotipo-fenotipo en el Particle Swarm Optimization.
  - El vector de velocidad $\mathbf{v}_i$ de un individuo i determina en que dirección se continuará la búsqueda y si esta tiene un carácter explorativo (alta velocidad) o explotativo (baja velocidad).

Particle Swarm Optimization (PSO)

- Los enlaces de información, se redefinen aleatoriamente en cada iteración: cada partícula informa a otras k partículas elegidas aleatoriamente (una partícula puede ser elegida en varios grupos).
- Cuando nos referimos a la mejor posición global nos referimos a la mejor posición encontrada en el grupo de los informantes de una partícula dada.

Particle Swarm Optimization (PSO)

- La versión básica del método PSO puede ser descrita mediante las siguientes ecuaciones:

$$
\begin{aligned}
v_i(t+1) &= av_i(t) + b_1 r_1 (p_i^{(1)} - x_i(t)) + b_2 r_2 (p_i^{(2)} - x_i(t)), \\
x_i(t+1) &= x_i(t) + v_i(k+1)
\end{aligned}
$$

  donde:

  - $v_i(t)$ denota la velocidad de la partícula i, que representa la distancia a ser recorrida por esta partícula desde su posición actual, esto es, la diferencia entre dos posiciones sucesivas de la partícula.
  - $x_i(t)$ representa la posición de la partícula.
  - $p_i^{(1)}$ representa su mejor posición previa.
  - $p_i^{(2)}$ es el mejor valor obtenido hasta el momento por cualquiera de las partículas vecinas.

  En la versión global del algoritmo, $p_i^{(2)}$ representa la mejor posición global entre todo el grupo.

Particle Swarm Optimization (PSO)

- Las partículas cambian su posición (o estado) de la siguiente forma:
  - En la iteración t, la velocidad $v_i(t+1)$ es actualizada en base a:

$$v_i(t+1) = av_i(t) + b_1 r_1 (p_i^{(1)} - x_i(t)) + b_2 r_2 (p_i^{(2)} - x_i(t)) \tag{32}$$

  - su valor actual está afectado por el parámetro de ajuste a,
  - y por un término que atrae a la partícula hacia posiciones mejores encontradas previamente.
  - La fuerza de la atracción es controlada por los coeficientes b_1 y b_2.
- La posición de la partícula $x_i(t+1)$ se actualiza en base a:
  - el valor actual de posición $x_i(t)$ y la velocidad recientemente calculada $v_i(t+1)$.

$$x_i(t+1) = x_i(t) + v_i(k+1) \tag{33}$$

Particle Swarm Optimization (PSO)

- Los tres parámetros de ajuste del algoritmo, a, b_1, and b_2, ejercen una gran influencia sobre las prestaciones que ofrece el algoritmo.
  - El peso del factor de inercia a es un parámetro especificado por el usuario, un factor de inercia grande presiona al algoritmo hacia exploraciones globales en nuevas áreas de búsqueda mientras que factores de inercia pequeños fuerzan al algoritmo a explorar más cuidadósamente el área de búsqueda actual.
  - Los coeficientes constantes y positivos de aceleración o factores de aprendizaje b_1 y b_2 controlan el tamaño del máximo paso de la partícula, son usuales valores $b_1 = b_2 = 2$.
  - Una adecuada selección de los factores de ajuste a, b_1, and b_2 permiten equilibrar la capacidad de búsqueda global (es decir, la exploración del espacio de estados), y local (es decir, la explotación del espacio de estados) del algoritmo.
  - Los números aleatorios r_1 and r_2 son seleccionados en el rango $[0, 1]$ e introducen una aleatoriedad útil en la explotación de la búsqueda en el espacio de estados.

Particle Swarm Optimization (PSO)

PSO Method

$t \rightarrow 0$ // iteration
Initialize a D-dimensional swarm, S
Evaluate fitness of each particle of swarm
while(termination criteria not met) {
  // generate a trial population
  for $(i = 1; i < S + 1; i + +)$ {
    for $(d = 1; i < D + 1; i + +)$ {
    Update velocity using basic velocity update equation;
    Update Position using basic position update equation;
    }
  Evaluate fitness of updated positions
  Update $p_i^{(1)}$ (p_i best) and $p_i^{(2)}$ (p best in S)
  }
$t \rightarrow t + 1$
}
⋮

Ant Colony Optimization (ACO)

- Ant Colony Optimization (ACO) es una técnica probabilística para la resolución de problemas de optimización que pueden reducirse a encontrar el mejor camino en un grafo.
- Se inspira en las trayectorias que describen las hormigas durante la búsqueda de comida, en su camino hasta el hormiguero.
- Es una algoritmo con una heurística constructiva.
- Inspirado por el experimento con hormigas reales de Goss et al. en 1989.
- Fue propuesto por Marco Dorigo en 1992 en su Tesis Doctoral.

Ant Colony Optimization (ACO)

Concepto natural

- En la naturaleza las hormigas vagan aleatoriamente (inicialmente) por el entorno a la búsqueda de comida y una vez que encuentran comida en la vuelta al hormiguero van depositando feromonas a lo largo del camino de vuelta.
- Si otras hormigas encuentran este camino (huelen las feromonas), ellas es probable que dejen de moverse aleatoriamente por el entorno, y que comiencen a seguir la pista, volviendo después y reforzando la pista química con nuevas feromonas si es que encuentran comida.
- Con el tiempo, la pista de feromonas comienza a evaporarse, reduciendo la fuerza de su atracción. Cuanto más tiempo le lleva a una hormiga el viajar a lo largo del camino y volver más tiempo se están evaporando las feromonas. Por un camino corto se viaja más frecuentemente que por otro mas largo y por tanto la densidad de feromonas es mayor en los caminos más cortos que en los más largos.
- La evaporación de las feromonas tiene la ventaja de evitar la convergencia hacia soluciones localmente óptimas. Si no hubiera evaporación el camino elegido por la primera hormiga tendería a ser excesivamente atractivo para las siguientes.

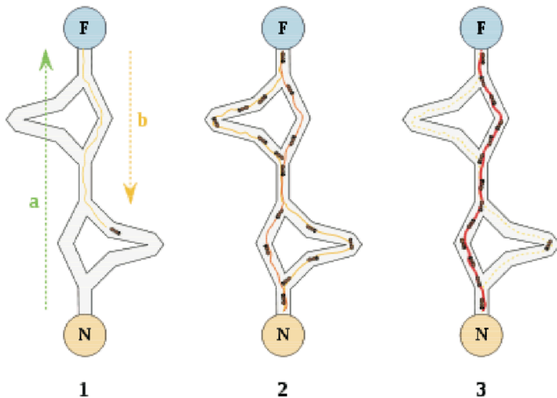
Ant Colony Optimization (ACO)

Conceptos generales

- Observando los caminos de ida y vuelta hasta el hormiguero que hacían las hormigas. Se vió que el camino de vuelta se hacía por la trayectoria más corta.
- Esto se produce debido a la liberación de feromonas por parte de las hormigas durante el camino, lo que hace que otras hormigas puedan posteriormente seguir el rastro de feromonas.
- Se comprobó empíricamente que tras un tiempo transitorio las hormigas seleccionan el camino más corto para llegar a la comida y volver.
- La probabilidad de seleccionar el camino más corto se acentúa más cuantas más diferencias hay entre varios caminos.

Ant Colony Optimization (ACO)

Experimento de Goss el al.



Ant Colony Optimization (ACO)

Algoritmo

```
Algoritmo    Optimización por Colonias de Hormigas
  {x_best:⟨hormiga⟩} = ACO(N: integer,F:⟨Trazo de Feromonas⟩)
var
  /*N es el número de hormigas*/
  /*F es el trazo inicial de feromonas*/
  C: ⟨Estructura de vecindad de una hormiga⟩;
  M: ⟨Estructura de memoria colectiva de las hormigas⟩;
  P: ⟨Estructura de probabilidad para cada movimiento de una hormiga⟩;
  pos: ⟨Posiciones de las hormigas⟩;

  begin
    repeat
      /*Construir una solución para cada hormiga*/
      for i := 1 to N do
        {M} := ActualizarMemoria(M);
        while not Solucionconstruida do
          {C} := MovimientosPosibles(pos); // Mtos. posibles de una hormiga
          {P} := EvaluarMovimientos(C,M); // Evaluar los movimientos
          {pos} := MovimientoHormiga(P); // Nuevo movimiento de la hormiga
          {F} := DepositarFeromonas(pos); // Depósito de feromonas
          {x} := AnnadirComponente(pos); // Añadir componente a la solución
        end while
        {x} = ActualizarSoluciones(x);
      end for
      {F} := EvaporacionFeromonas(x,F);
      {F} := AccionesDemonio(x,F);
    until terminacion
    {x_best} := SeleccionarMejor(x); //Seleccionar la mejor solución encontrada
  end
```

Ant Colony Optimization (ACO)

Elección del siguiente vértice en el recorrido

$$p_k(r,s) = \begin{cases} \dfrac{[\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}{\sum\limits_{u \in J_k(r)} [\tau_{ru}]^{\alpha} \cdot [\eta_{ru}]^{\beta}}, & \text{si } s \in J_k(r) \\ 0, & \text{en otro caso} \end{cases}$$

- $p_k(r,s)$: probabilidad de que la hormiga k, estando en r, elija irse al vértice s, es decir que recorra la arista a_{rs}.
- τ_{rs}: feromona de la arista a_{rs}.
- $J_k(r)$: conjunto de vértices alcanzables desde r no visitados por la hormiga k.
- η_{rs}: información heurística de la arista a_{rs} (visibilidad: inverso de la distancia).
- α, β: pesos que establecen un equilibrio entre la importancia de las informaciones memorística y heurística respectivamente.

Ant Colony Optimization (ACO)

Actualización de feronomas

$$\tau_{rs}(t) = (1-\rho)\cdot\tau_{rs}(t-1) + \sum_{k=1}^{m} \Delta\tau_{rs}^{k}$$

- $\tau_{rs}(t)$: feromona del arco a_{rs} en el instante t (nuevo valor).
- $\tau_{rs}(t-1)$: feromona del arco a_{rs} en el instante $t-1$ (valor actual).
- p: factor de evaporación de feromonas.
- m: número de hormigas.
- $\Delta\tau_{rs}^{k}$: aportación de feromonas de la hormiga k:
  - Una cantidad constante o ponderada si la hormiga k ha visitado la arista a_{rs}.
  - 0 en caso contrario.

Ant Colony Optimization (ACO)

- Algunas características del algoritmo:
  - Se emplean hormigas artificiales que iterativamente añaden una componente a la solución teniendo en cuenta:
    - Información heurística.
    - Rastros de feromonas que cambian dinámicamente (memoria).
  - Ventajas:
    - La componente estocástica de ACO hace que las hormigas construyan una amplia variedad de soluciones.
    - La información heurística guía a las hormigas hacia soluciones mejores.
    - La experiencia de las hormigas se puede utilizar para construir soluciones mejores en futuras iteraciones.

Ant Colony Optimization (ACO)

- Algunos parámetros de búsqueda:
  - Factor de aportación de feromonas. Realimentación positiva, cuanto mejor sea la solución, más feromonas se aportan en el camino.
  - Factor de evaporación de feromonas. Permite olvidar las malas decisiones (estancamiento local) producidas en el pasado.
  - Pesos que establecen un equilibrio entre decisiones heurísticas y basadas en feromononas.
  - Cantidad de feromona inicial de cada arista.

Bibliografía

- Goldberg: Genetic Algorithms in Search, Optimization and Machine Learning
- Mitchell: An Introduction to Genetic Algorithms
- Davis: Handbook of Genetic Algorithms
- Haupt and Haupt: Practical Genetic Algorithms
- Gen and Cheng [787]: Genetic Algorithms and Engineering Design
- Holland: Adaptation in Natural and Artificial Systems