Metaheurísticas

Práctica 2. Metaheurísticas basadas en un única solución

Pedro Antonio Gutiérrez

Asignatura "Metaheurísticas"
3º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

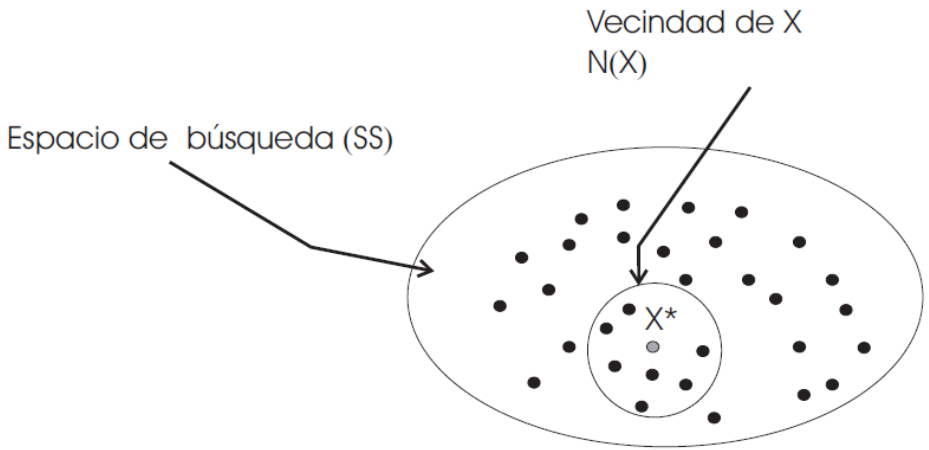
15 de marzo de 2014

Objetivos de la práctica

- Familiarizar al alumno con los conceptos vecindad, movimientos...
- Introducir las metaheurísticas basadas en una única solución y aplicarlas a dos de los problemas estudiados durante la primera práctica.
- En concreto, se van a considerar las metaheurísticas:
  - Búsqueda local.
  - Enfriamiento simulado.

Entorno/vecindario

- Dada un espacio de búsqueda SS para un problema concreto, que viene definido por el esquema de representación de soluciones considerado, y dada una solución $\mathbf{x} \in SS$, el entorno/vecindario $N(\mathbf{x})$ de esa solución es un subconjunto del espacio de soluciones que contiene aquellas soluciones que están "próximas" a la solución considerada:



Entorno/vecindario: posible definición (fuerza bruta)

- Dado dicho espacio SS, se puede definir una función de distancia $dist(\mathbf{x}, \mathbf{y})$ entre cualquier par de soluciones $\mathbf{x}$ y $\mathbf{y}$:

$$dist : SS \times SS \rightarrow \mathbb{R} \tag{1}$$

- De esta forma, podemos definir matemáticamente el vecindario de $\mathbf{x}$, $N(\mathbf{x})$:

$$N(\mathbf{x}) = \{\mathbf{y} \in SS : dist(\mathbf{x}, \mathbf{y}) < \epsilon\} \tag{2}$$

  done ϵ marca la amplitud del vecindario considerado.
- Ejemplos:
  - SS espacio euclídeo $\rightarrow$ distancia euclídea.
  - SS espacio binario $\rightarrow$ distancia *Hamming*.

Movimientos

- En el entorno $N(\mathbf{x})$ de una solución $\mathbf{x}$, se encuentran todas aquellas soluciones $\mathbf{y} \in SS$ accesibles desde $\mathbf{x}$ a través de una operación llamada movimiento/operador de generación de vecino.
- Dada una solución $\mathbf{x}$, se puede acceder a cada solución de su entorno $\mathbf{y} \in N(\mathbf{x})$, aplicando el operador de generación de vecino.
- Algoritmos de búsqueda por entorno/trayectorias simples/basados en única solución:
  - Idea básica: manejar una única solución en SS y "moverse" a otras "soluciones vecinas".

Entornos grandes

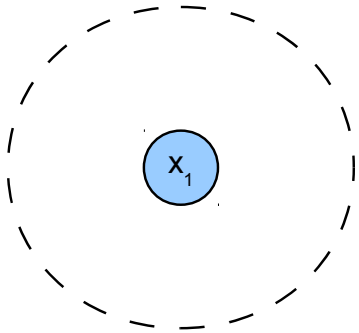
- En algunas ocasiones puede no ser recomendable explorar el entorno al completo.
  - Tamaño muy grande (p.e., instancias muy grandes o entorno exponencial con respecto al tamaño del problema).
  - Muchas soluciones vecinas presentan el mismo valor objetivo que la solución actual (espacio objetivo "plano").
  - Solución: aplicar exploración parcial del entorno. Considerando información específica del problema (heurística) podemos:
    - Restringir el entorno de una solución para incluir únicamente movimientos prometedores, llamados listas de candidatos (reducir el entorno).
    - Definir un orden sistemático de exploración que considere primero la aplicación de los movimientos más prometedores (explorar el entorno con preferencias).

Factorización

- Las soluciones pertenecientes a un entorno son muy parecidas entre sí, los operadores de vecino provocan cambios "suaves".
- Esto hace que, cuando se aplica un movimiento sobre una solución $\mathbf{x}$ para obtener una solución vecina $\mathbf{x}'$:
  - Debamos evitar calcular el valor de la función objetivo $\mathbf{x}'$ desde cero (método original).
  - En su lugar, siempre que sea posible, debemos aplicar un mecanismo denominado factorización para calcular el valor objetivo de $\mathbf{x}'$ a partir del de $\mathbf{x}$ considerando sólo los cambios realizados por el operador de movimiento.

Búsqueda local

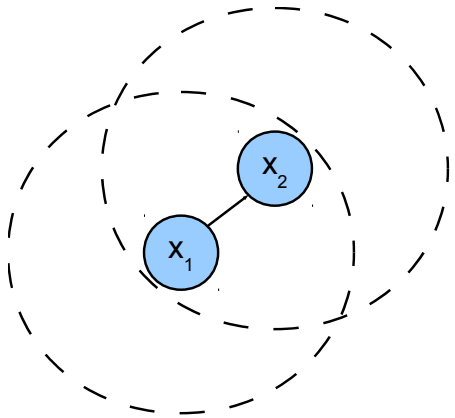
- Búsqueda local: Procedimiento iterativo de búsqueda donde, dada una solución actual, seleccionamos una solución de su entorno para continuar la búsqueda.



SoluciónInicial

Búsqueda local

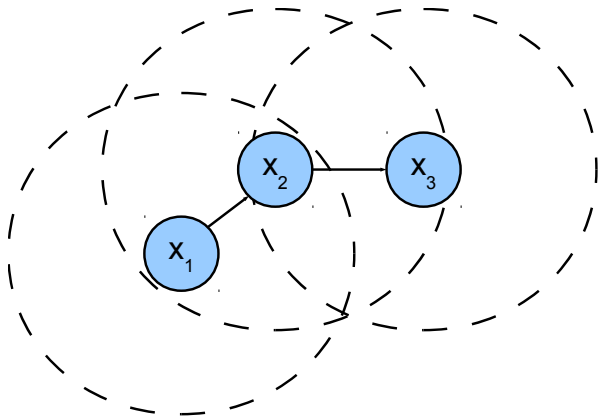
- Búsqueda local: Procedimiento iterativo de búsqueda donde, dada una solución actual, seleccionamos una solución de su entorno para continuar la búsqueda.



SoluciónInicial

Búsqueda local

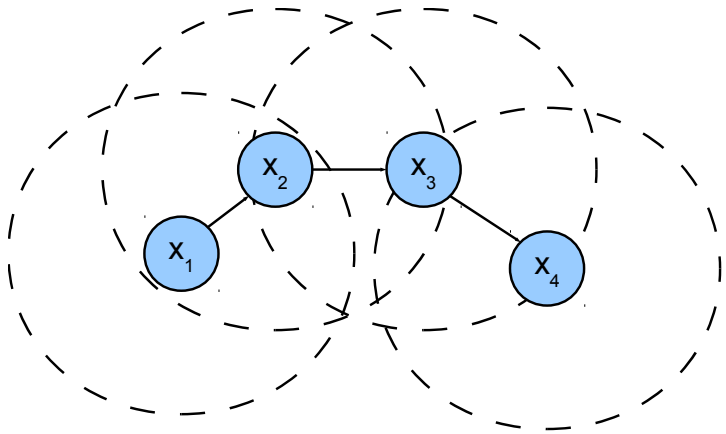
- Búsqueda local: Procedimiento iterativo de búsqueda donde, dada una solución actual, seleccionamos una solución de su entorno para continuar la búsqueda.



SoluciónInicial

Búsqueda local

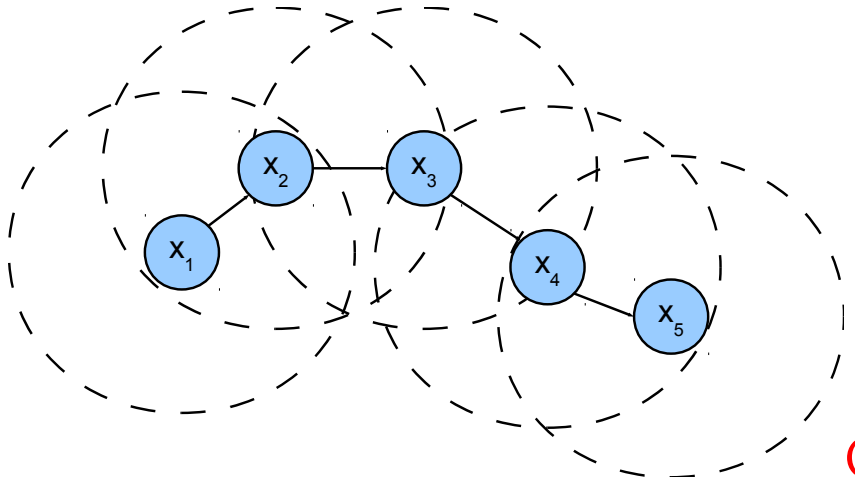
- Búsqueda local: Procedimiento iterativo de búsqueda donde, dada una solución actual, seleccionamos una solución de su entorno para continuar la búsqueda.



SoluciónInicial

Búsqueda local

- Búsqueda local: Procedimiento iterativo de búsqueda donde, dada una solución actual, seleccionamos una solución de su entorno para continuar la búsqueda.



SoluciónInicial

Óptimo ¿global?

Búsqueda aleatoria pura y por recorrido al azar

- Búsqueda aleatoria pura: lo que hicimos en la primera práctica.
  - Generar n soluciones, evaluar cada una de ellas y escoger la mejor.
- Búsqueda aleatoria por recorrido al azar:
  - Generar una solución inicial aleatoria.
  - Hasta que se cumpla criterio de parada:
    - Generar el entorno de la solución actual.
    - Escoger una solución de dicho entorno de forma aleatoria.

Búsqueda local: pseudocódigo

Procedimiento general de búsqueda local

Inicio

1. SolucionActual ← generaSolucionAleatoria()
2. **Repetir**
   1. Vecindario ← generaVecindario(SolucionActual)
   2. SolucionNueva ← eligeSolucion(Vecindario)
   3. **Si** (valorObjetivo(SolucionNueva) "mejor que" valorObjetivo(SolucionActual))
      SolucionActual ← SolucionNueva

   Hasta (CondicionParada)
3. **Devolver** SolucionActual

Fin

Búsqueda local: variantes

- Dos opciones:
  - Búsqueda local del mejor (BLb):
    - eligeSolucion(Vecindario): escoge la mejor solución del vecindario, es decir, evalúa todas las soluciones y coge la que obtiene el mejor valor objetivo.
    - También llamado *best improvement*.
    - Computacionalmente costoso.
  - Búsqueda local del primer mejor (BLf):
    - eligeSolucion(Vecindario): escoge a la primera solución del vecindario que mejora algo la solución actual.
    - También llamado *first improvement*.
    - Computacionalmente menos costoso.
    - Nos podemos olvidar de alguna solución que era mejor que la escogida.

Búsqueda local del mejor: pseudocódigo

Procedimiento búsqueda local del mejor (BLb)

Inicio

1. SolucionActual ← generaSolucionAleatoria()
2. **Repetir**
   1. Vecindario ← generaVecindario(SolucionActual)
   2. MejorObjetivo ← valorObjetivo(SolucionActual)
   3. **Repetir** para cada Vecino de Vecindario:
      1. **Si** (valorObjetivo(Vecino) "mejor que" MejorObjetivo)
         SolucionActual ← Vecino
         MejorObjetivo ← valorObjetivo(Vecino)

   Hasta (CondicionParada OR NoVecinoMejor)
3. **Devolver** SolucionActual

Fin

Búsqueda local del primer mejor: pseudocódigo

Procedimiento búsqueda local del primer mejor (BLf)

Inicio

1. SolucionActual ← generaSolucionAleatoria()
2. **Repetir**
   1. Vecindario ← generaVecindario(SolucionActual)
   2. ValorActual ← valorObjetivo(SolucionActual)
   3. **Repetir**
      1. Vecino ← escogerAleatoriamenteSinRepeticion(Vecino)

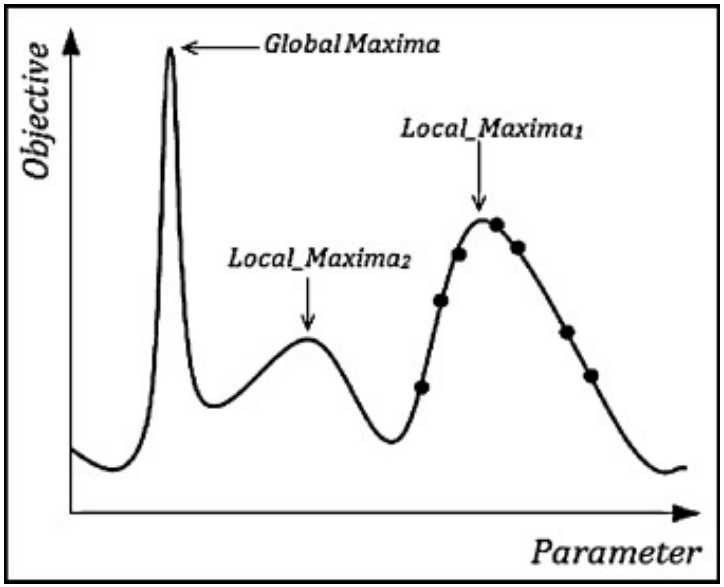
      Hasta (valorObjetivo(Vecino) "mejor que" ValorActual OR NoVecinoMejor)
   4. SolucionActual ← Vecino

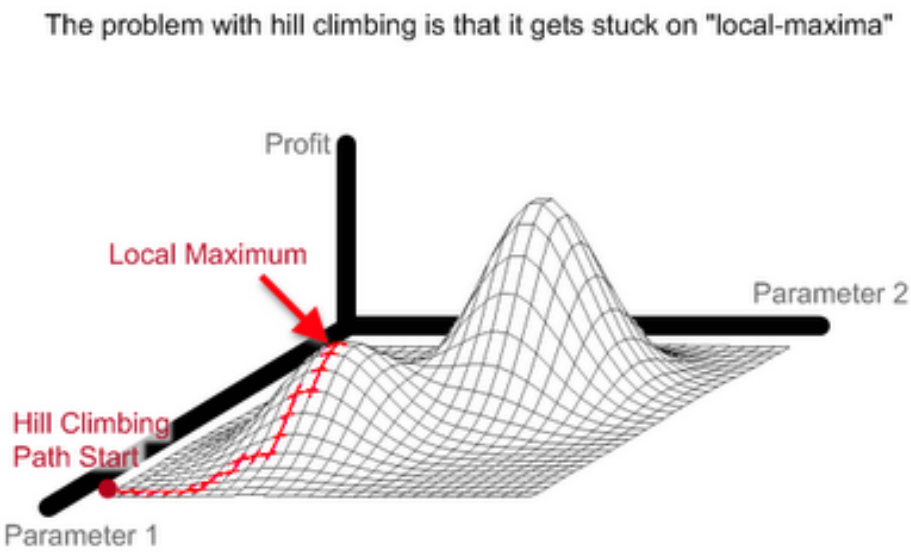
   Hasta (CondicionParada OR NoVecinoMejor)
3. **Devolver** SolucionActual

Fin

Búsqueda local: limitaciones

- Suele caer en óptimos locales, que a veces están bastante alejados del óptimo global:





Búsqueda local: limitaciones

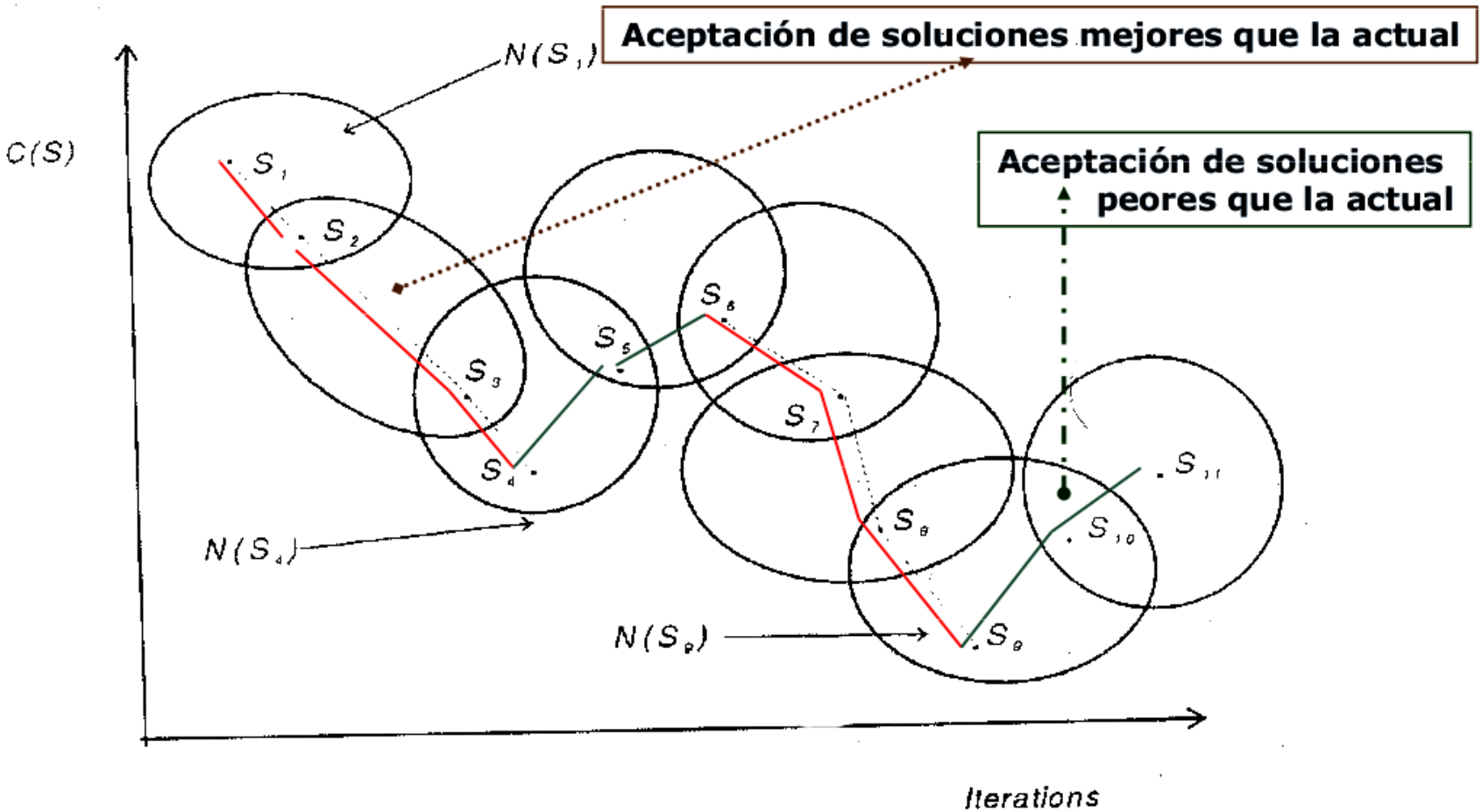
- ¿Cómo se puede resolver?
  - Permitir movimientos de empeoramiento de la solución actual: enfriamiento simulado, búsqueda tabú...
  - Modificar la estructura de entornos: búsqueda tabú, búsqueda en entornos variables (VNS)...
  - Volver a comenzar la búsqueda desde otra solución inicial: búsquedas multiarranque, VNS...
- Nunca generar todo el vecindario de una solución, sino solo la solución que me interesa.

Enfriamiento simulado

- El Enfriamiento o Recocido Simulado es un algoritmo de búsqueda por entornos con un criterio probabilístico de aceptación de soluciones basado en Termodinámica.
  - Permite que algunos movimientos sean hacia soluciones peores, para evitar que se finalice en óptimos locales.
  - Sin embargo, hay que controlar estos "*movimientos de escape*" porque puede que estemos camino de la mejor solución global.
  - Filosofía habitual: diversificar al principio e intensificar al final.
  - El ES controla la frecuencia de los "*movimientos de escape*" mediante una función de probabilidad que hará disminuir la probabilidad de éstos movimientos hacia soluciones peores conforme avanza la búsqueda (y por tanto estamos más cerca, previsiblemente, del óptimo global).

Enfriamiento simulado

Se aceptan soluciones peores que la actual (pero siempre hay que guardar la mejor solución encontrada hasta el momento):



Enfriamiento simulado: fundamentos del algoritmo

- El ES es un método de búsqueda por entornos caracterizado por un criterio de aceptación de soluciones vecinas que se adapta a lo largo de su ejecución.
- Hace uso de una variable llamada Temperatura, T, cuyo valor determina en qué medida pueden ser aceptadas soluciones vecinas peores que la actual.
  - T se inicializa a un valor alto, denominado Temperatura inicial, T_0, y se va reduciendo en cada iteración mediante un mecanismo de enfriamiento de la temperatura, $Enfria(T)$, hasta alcanzar una Temperatura final, T_f.

Enfriamiento simulado: fundamentos del algoritmo

- En cada iteración (enfriamiento) se genera un número concreto de vecinos en un bucle interno.
- Por cada vecino generado en el bucle interno:
  - Si mejora el valor de la función objetivo, la solución generada sustituye a la actual.
  - Si empeora, aún existe la probabilidad de que el vecino sustituya a la solución actual. Esto permite al algoritmo salir de óptimos locales, en los que la BL clásica quedaría atrapada.
- El bucle externo termina cuando se alcanza la temperatura final, $T \leq T_f$, o cuando se cumple algún otro criterio de parada.

Enfriamiento simulado: fundamentos del algoritmo

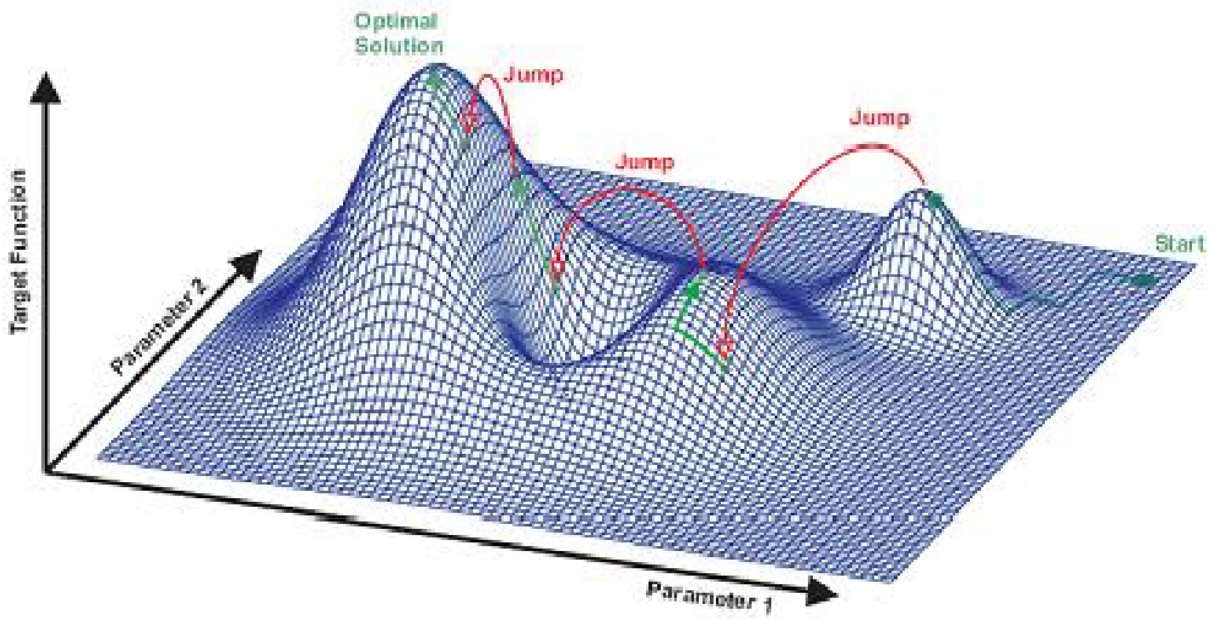
- Lo más importante es la probabilidad de aceptación de soluciones peores, que depende de la diferencia del valor objetivo de la solución actual y la vecina, Δf, y de T:

$$p_{\text{Aceptacion}} = \exp\left(-\frac{\Delta f}{T}\right) = \frac{1}{\exp\left(\frac{\Delta f}{T}\right)} \tag{3}$$

  - A mayor temperatura, $T \uparrow$, mayor probabilidad de aceptación, $p_{\text{Aceptacion}} \uparrow$. Así, el algoritmo acepta más soluciones peores que la actual al principio de la ejecución (exploración) pero no al final (explotación).
  - A menor diferencia de valores objetivos, $\Delta f \downarrow$, mejor es la solución a aceptar, por tanto, mayor es su probabilidad de aceptación, $p_{\text{Aceptacion}} \uparrow$.

Enfriamiento simulado: fundamentos del algoritmo

- Δf debería ser mayor cuanto peor es la solución vecina generada:
  - Problema de minimización: $\Delta f = f' - f$.
  - Problema de maximización: $\Delta f = f - f'$.
- Estos movimientos de escape suponen un salto en la búsqueda:



Enfriamiento simulado: pseudocódigo (minimización)

Procedimiento Enfriamiento Simulado (ES)

Inicio

1. $T \leftarrow T_0$; SolucionActual $\leftarrow$ generaSolucionAleatoria(); MejorSolucion $\leftarrow$ SolucionActual;
2. **Repetir** */* Bucle externo */*
   1. **Repetir** */* Bucle interno */*
      1. Vecino $\leftarrow$ nuevoVecino(SolucionActual)
      2. $\Delta f \leftarrow$(valorObjetivo(Vecino)$-$valorObjetivo(SolucionActual))
      3. **Si** ($\Delta f < 0$ OR $U(0,1) \leq \exp(-\Delta f / T)$)
         SolucionActual $\leftarrow$ Vecino
         Si (valorObjetivo(Vecino) $<$ valorObjetivo(MejorSolucion))
         MejorSolucion $\leftarrow$ Vecino

      Hasta (CondicionParadaBucleInterno)
   2. $T \leftarrow Enfria(T)$ */* Esquema de enfriamiento */*

   Hasta (CondicionParadaBucleExterno OR $T \leq T_f$)
3. **Devolver** MejorSolucion

Fin

Enfriamiento simulado: pseudocódigo (maximización)

Procedimiento Enfriamiento Simulado (ES)

Inicio

1. $T \leftarrow T_0$; SolucionActual $\leftarrow$ generaSolucionAleatoria(); MejorSolucion $\leftarrow$ SolucionActual;
2. **Repetir** */* Bucle externo */*
   1. **Repetir** */* Bucle interno */*
      1. Vecino $\leftarrow$ nuevoVecino(SolucionActual)
      2. $\Delta f \leftarrow$(valorObjetivo(SolucionActual)$-$valorObjetivo(Vecino))
      3. **Si** ($\Delta f < 0$ OR $U(0,1) \leq \exp(-\Delta f / T)$)
         SolucionActual $\leftarrow$ Vecino
         Si (valorObjetivo(Vecino) $>$ valorObjetivo(MejorSolucion))
         MejorSolucion $\leftarrow$ Vecino

      Hasta (CondicionParadaBucleInterno)
   2. $T \leftarrow Enfria(T)$ */* Esquema de enfriamiento */*

   Hasta (CondicionParadaBucleExterno OR $T \leq T_f$)
3. **Devolver** MejorSolucion

Fin

Enfriamiento simulado: pseudocódigo

- Muchos esquemas de enfriamiento:
  - El más frecuente: enfriamiento geométrico, $T_{k+1} = T_k \cdot \alpha$, con $0 < \alpha < 1$.
- Temperatura inicial T_0:
  - No parece sensato hacerla independiente del problema.
  - Alternativa:

$$T_0 = \frac{\nu \cdot z(\mathbf{x}_0)}{-\ln(\mu)} \tag{4}$$

siendo $z(\mathbf{x}_0)$ el valor objetivo de la solución aleatoria inicial y $\mu \in [0, 1]$ la probabilidad de aceptar una solución un ν por uno distinta a la inicial.

Problema MMDP

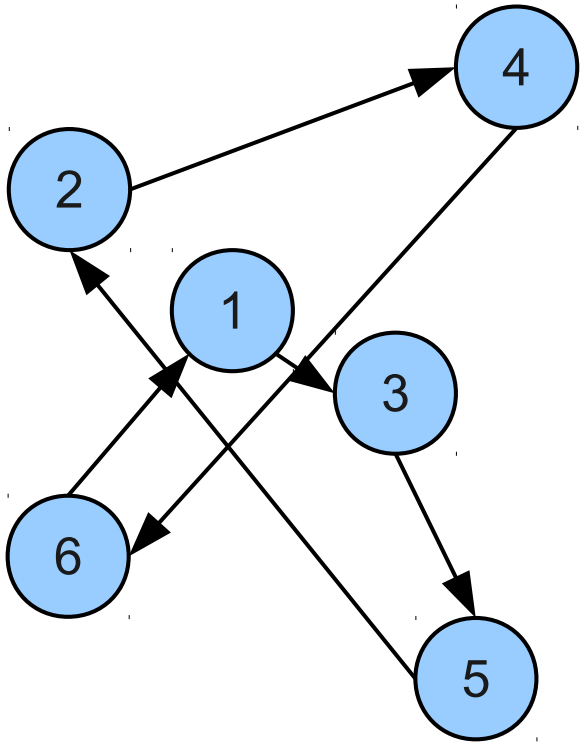
- Esquema de generación de vecinos: dada una solución **x** donde el elemento i-ésimo está seleccionado ($x_i = 1$) y el elemento j-ésimo no está seleccionado ($x_j = 0$), el movimiento $Intercambio(\mathbf{x}, i, j)$ deselecciona el elemento i-ésimo y selecciona el elemento j-ésimo.
  - Si el número total de elementos es n y el número de elementos a seleccionar es m, el tamaño del vecindario es $m \cdot (n - m)$ (número de unos por número de ceros).
  - Para algunos problemas el tamaño es $n = 500$, $m = 50 \rightarrow m \cdot (n - m) = 50 \cdot 450 = 22,500$.
    - No generar el vecindario completo, únicamente la solución que vamos a evaluar.
    - Podemos generar una lista de índices de vecinos de 1 a $m \cdot (n - m)$ y luego seleccionar la solución que me interesa.

Problema TSP

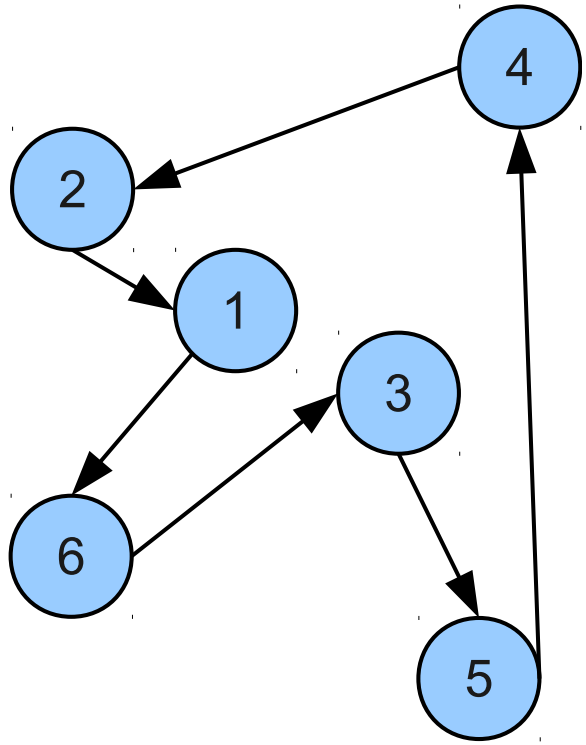
- Esquema de representación de las soluciones: se seguirá la representación en forma de permutación **x** de los identificadores de las ciudades (es decir, x_i será el identificador de la ciudad que se visita en la posición i-ésima al realizar el recorrido).

$$\text{Minimizar:} \quad z_{\text{TSP}}(\mathbf{x}) = \sum_{i=1}^{m-1} d_{x_i x_{i+1}} + d_{x_n x_1}. \qquad (5)$$

Problema TSP



$$x = \{1,3,5,2,4,6\}$$



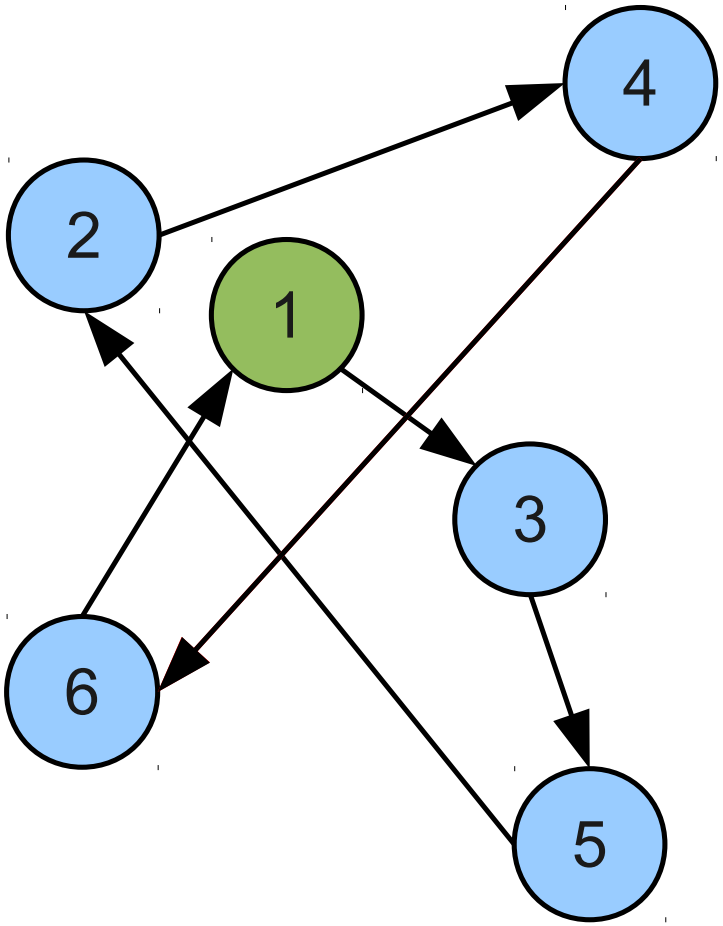
$$x = \{1,6,3,5,4,2\}$$

Problema TSP: esquema generación de vecinos

- Se puede usar cualquier operador de vecino para permutaciones.
- Consideraremos el operador de intercambio de dos aristas del recorrido, $2opt$. Sean i y j el índice de dos aristas del recorrido.

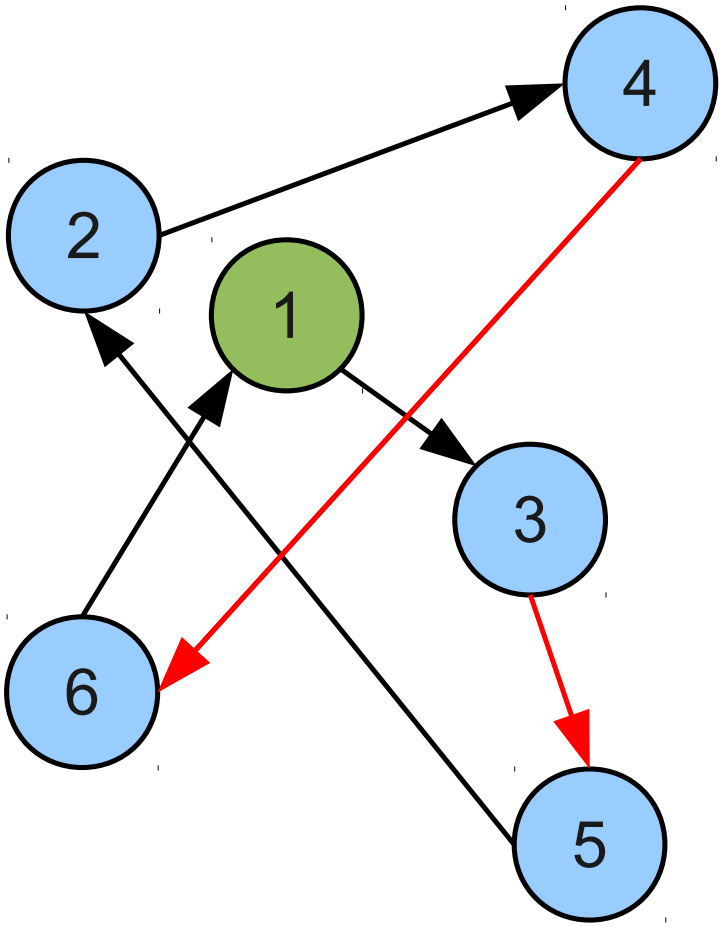
Problema TSP: esquema generación de vecinos

- $\mathbf{x} = \{1, 3, 5, 2, 4, 6\}$, $2opt(\mathbf{x}, 1, 4) = \{1, 3, 4, 2, 5, 6\}$
- Movimiento a aplicar: intercambiar aristas 1 y 4:



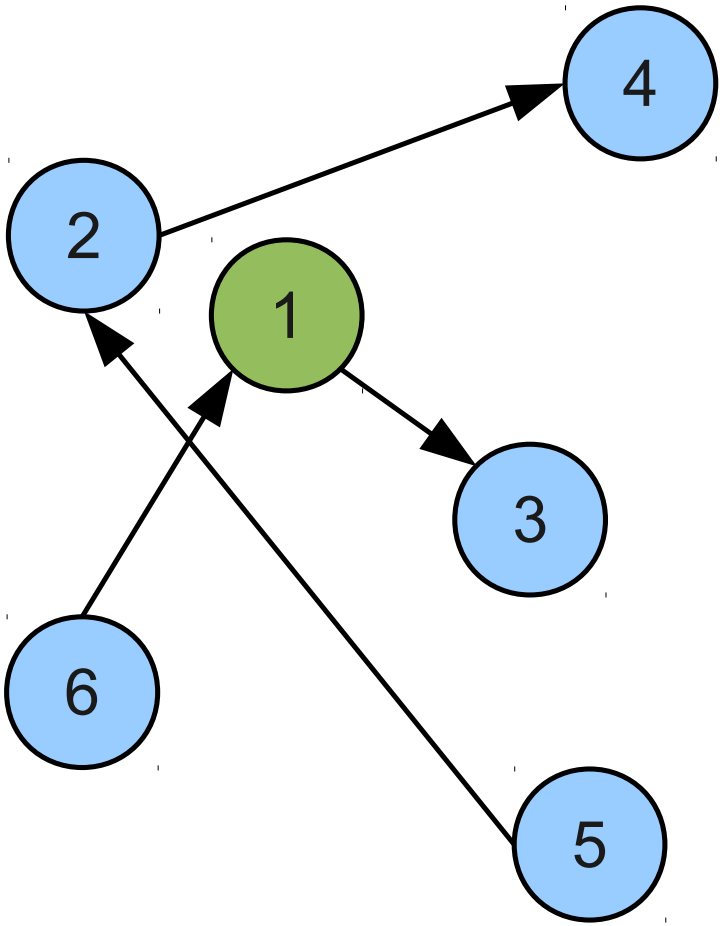
Problema TSP: esquema generación de vecinos

- $\mathbf{x} = \{1, 3, 5, 2, 4, 6\}$, $2opt(\mathbf{x}, 1, 4) = \{1, 3, 4, 2, 5, 6\}$
- Seleccionamos las aristas:



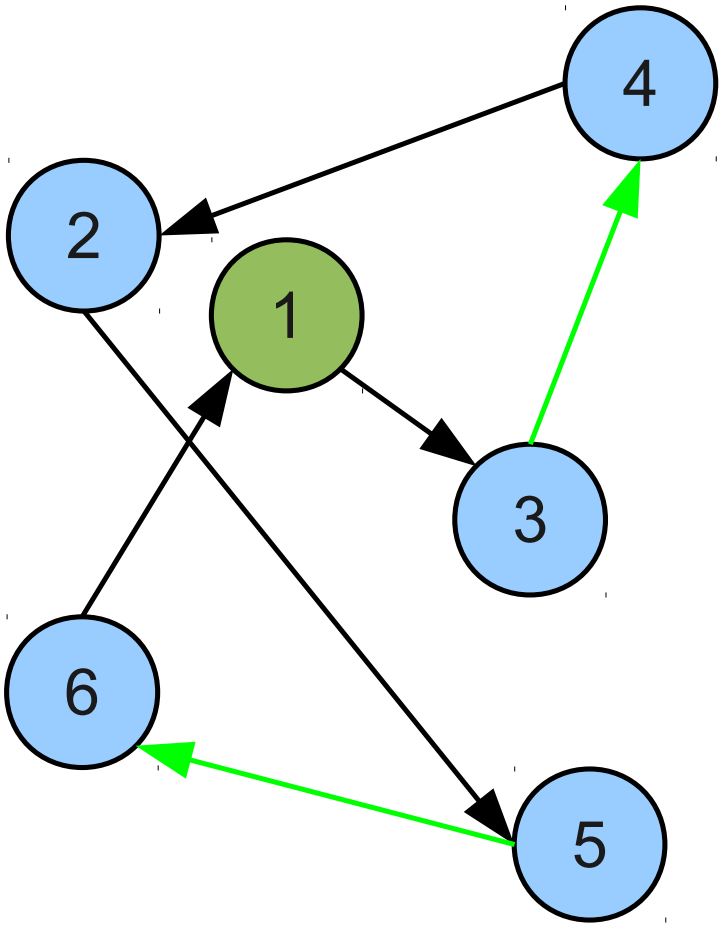
Problema TSP: esquema generación de vecinos

- $\mathbf{x} = \{1, 3, 5, 2, 4, 6\}$, $2opt(\mathbf{x}, 1, 4) = \{1, 3, 4, 2, 5, 6\}$
- Las eliminamos:



Problema TSP: esquema generación de vecinos

- $\mathbf{x} = \{1, 3, 5, 2, 4, 6\}$, $2opt(\mathbf{x}, 1, 4) = \{1, 3, 4, 2, 5, 6\}$
- Reconectamos las aristas:



Problema TSP: esquema generación de vecinos

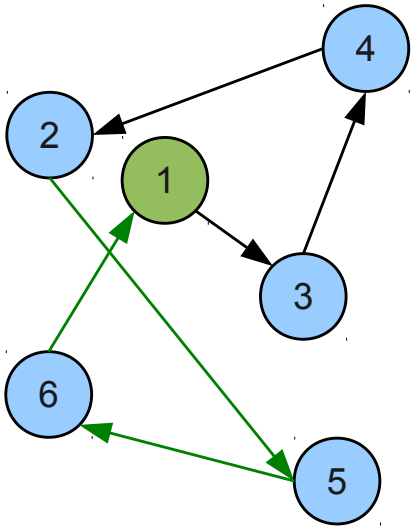
- $\mathbf{x} = \{1, 3, 5, 2, 4, 6\}$, $2opt(\mathbf{x}, 1, 4) = \{1, 3, 4, 2, 5, 6\}$
- Al reconectar las aristas solo hay una forma de hacerlo.
- Algunas cambian de sentido ($4 \rightarrow 2$ y $2 \rightarrow 5$), aunque nos da igual ya que el TSP es simétrico.
- $2opt(\mathbf{x}, i, j)$: Si suponemos que $i < j$, el movimiento se resume en intercambiar el destino de la primera arista con el origen de la segunda, es decir, intercambiar las posiciones x_{i+1} y x_j.
- Dada una arista i, dicha arista solo se puede intercambiar con aristas que no compartan ninguna ciudad con ella $\rightarrow$ Si fijamos i, hay $m - 3$ aristas j en las que tenga sentido aplicar $2opt(\mathbf{x}, i, j)$ (todas menos la anterior, la posterior y ella misma).

Problema TSP: lista de candidatos (CV)

- El objetivo del TSP consiste en minimizar el recorrido indicado por **x**.
- Por tanto, conviene eliminarnos del recorrido aquellas aristas cuya distancia sea muy grande.
- La lista de aristas candidatas CV será una vector de tamaño $\beta \cdot m$ que incluya a aquellas aristas con mayor distancia.
- Para construir la lista, ordenaremos las aristas por distancia en orden decreciente y seleccionaremos las $\beta \cdot m$ primeras (redondeando hacia arriba).
- Vamos a tomar $\beta = 0,5$.

Problema TSP: lista de candidatos (CV)

$\mathbf{x} = \{1, 3, 5, 2, 4, 6\}$



Supongamos $\{d_{13}, d_{35}, d_{52}, d_{24}, d_{46}, d_{61}\} = \{2{,}03{,}04{,}05{,}04{,}14{,}2\}$
Si $\beta = 0,5$, $CV = \{3, 4, 5\}$ (las tres aristas de mayor tamaño).

Problema TSP: lista de candidatos (CV)

- CV define el conjunto de aristas que se van a intentar mover.
  - Para los algoritmos BL, se va a explorar en orden descendiente de distancia.
  - Para el ES, se explora aleatoriamente.
  - Una vez escogido una arista i de CV, todavía quedaría escoger la arista j:
    - Como ya hemos comentado, solo hay $m - 3$ aristas posibles.
    - Para BLb, las recorremos todas.
    - Para BLf, las vamos recorriendo en orden aleatorio sin repetición.
    - Para ES, escogemos aleatoriamente una (puede haber repetición).

Problema TSP: factorización

- Para reducir el coste computacional, en lugar de calcular desde cero la distancia de la solución vecina obtenida, podemos expresarla como una función de la distancia original.
- Añadiremos la distancia de las aristas nuevas y eliminaremos la distancia de las aristas que han desaparecido.
- Suponemos $i < j$ y que $\mathbf{x}' = 2opt(\mathbf{x}, i, j)$:

$$\begin{aligned} z_{\mathrm{TSP}}(\mathbf{x}') &= z_{\mathrm{TSP}}(\mathbf{x}) + \\ &+ \left(d_{\mathbf{x}'_i,\mathbf{x}'_{i+1}} + d_{\mathbf{x}'_{i+1},\mathbf{x}'_{i+2}} + d_{\mathbf{x}'_{j-1},\mathbf{x}'_j} + d_{\mathbf{x}'_j,\mathbf{x}'_{j+1}}\right) - \\ &- \left(d_{\mathbf{x}_{j-1},\mathbf{x}_j} + d_{\mathbf{x}_j,\mathbf{x}_{j+1}} + d_{\mathbf{x}_i,\mathbf{x}_{i+1}} + d_{\mathbf{x}_{i+1},\mathbf{x}_{i+2}}\right) \end{aligned}$$

- $j + 1$ debe hacerse en módulo m.

Metaheurísticas

Práctica 2. Metaheurísticas basadas en un única solución

Pedro Antonio Gutiérrez

Asignatura "Metaheurísticas"
3º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

15 de marzo de 2014