Tema 3: Metaheurísticas de Mejora Iterativa (Búsqueda Local)
Iterative Improvement (Local Search)

Enfriamiento Simulado (Simulated Annealing)

Búsqueda Tabú (Tabu Search)

Búsqueda por Haz (Beam Search)

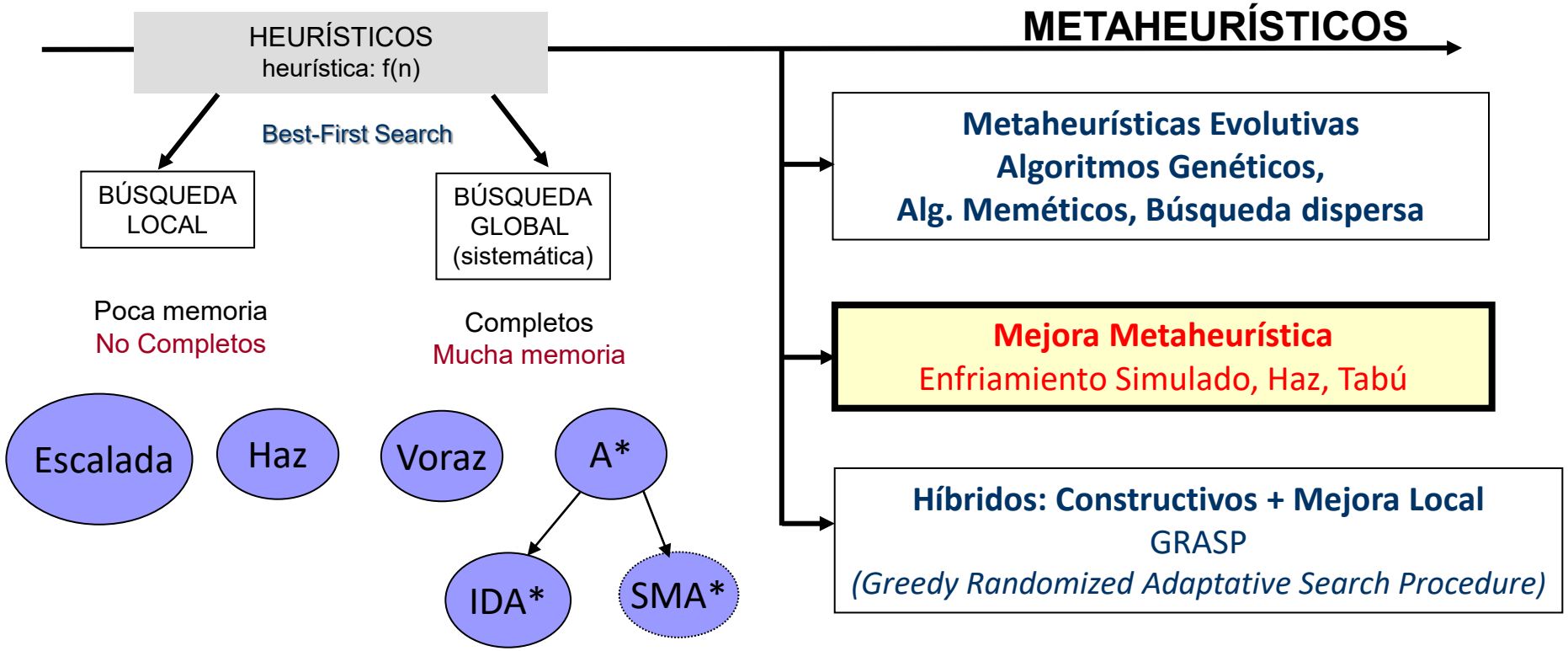
Prácticas

Aplicación metaheurísticas – **Enfriamiento Simulado** (2 semanas)

Bibliografía

- **Monografía: Metaheurísticas. Inteligencia Artificial, Vol 7, No 19 (journal.iberamia.org) (2003).**
- **Inteligencia Artificial. Técnicas, métodos y aplicaciones (cap. 9).** Palma, Marín (eds). Mc Graw Hill (2008).
- **An Introduction to Metaheuristics for Optimization, B. Chopard, M. Tomassini. Springer (2018).**

- **Artículos en Poliformat. Detalles de metaheurísticas, Estudios comparativos**



Los métodos de búsqueda local consumen poca memoria, pero suelen caer en óptimos locales, pudiendo incluso revisitar estados (ciclos)

Hay que poder escapar de estos óptimos locales

$\Rightarrow$ Metaheurísticas *más permisivas*

Metaheurísticas de Mejora (búsqueda local)

Idea: Mejora iterativa de la solución mediante **búsqueda local (y limitada) en el entorno** pasando de una solución a otra vecina mejor (algoritmos de escalada)

Base: Escalada (hill-climbing)

- Se parte de una solución inicial.
- Se obtienen sus "vecinos", en una búsqueda en la vecindad
- Se sustituye la solución por **un mejor vecino**, si cumple el criterio de aceptación
- Se repiten los pasos anteriores mientras se satisfaga el criterio de continuación
- El proceso se acaba cuando no hay mejora posible en el conjunto de soluciones vecinas (valles, llanuras, ciclos, etc.)

Esquema general (minimización):

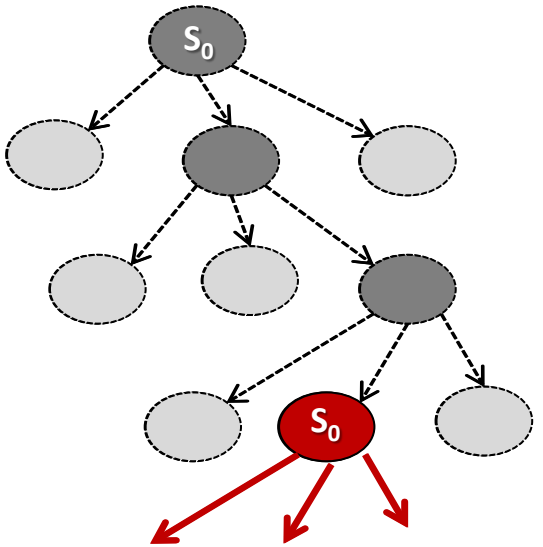
Generar una solución inicial $s_0 \in S$

Repetir:

Seleccionar $s \in N(s_0)$, tal que $f(s) < f(s_0)$

Reemplazar s_0 por s

hasta que $\forall s \in N(s_0)$, $f(s) \geq f(s_0)$



Vecindario N(s0)

Ejemplo. Viajante de comercio (14 ciudades)

Inicial: (A, B, C, D, E, F, G, H, I, J, K, L, M, N, A)

Entorno: Intercambiar ?x por ?y

(14 * 13 = 182 vecinos,
de 14!=10^{11} soluciones)

Esquema general (minimización)

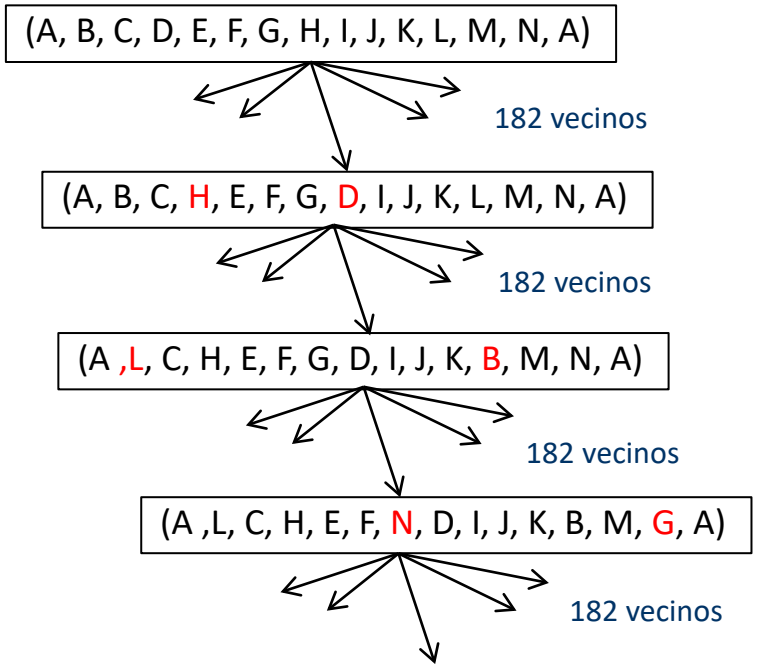
Generar una solución inicial $s_0 \in S$

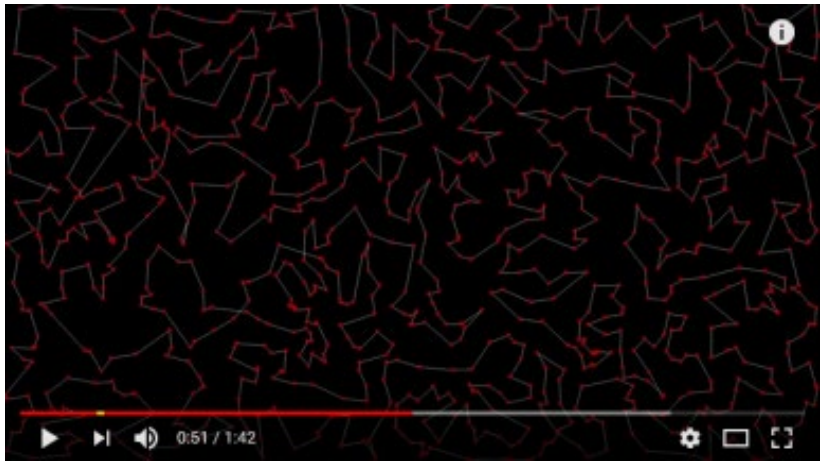
Repetir:

→ Seleccionar $s \in N(s_0)$ / $f(s) < f(s_0)$

Reemplazar s_0 por s

hasta que $\forall s \in N(s_0)$, $f(s) \geq f(s_0)$





¿Y si ninguna solución vecina mejora?

En resumen, la Escalada…

- Recorre un espacio de soluciones transformando iterativamente una solución de partida
- Requiere:
  - Una **solución inicial** $s_0 \in S$, y una función de evaluación de las soluciones f(s)
  - Definir una **estructura de entorno** *N*, $\forall s \in S$, define $N(s) \subseteq S$.

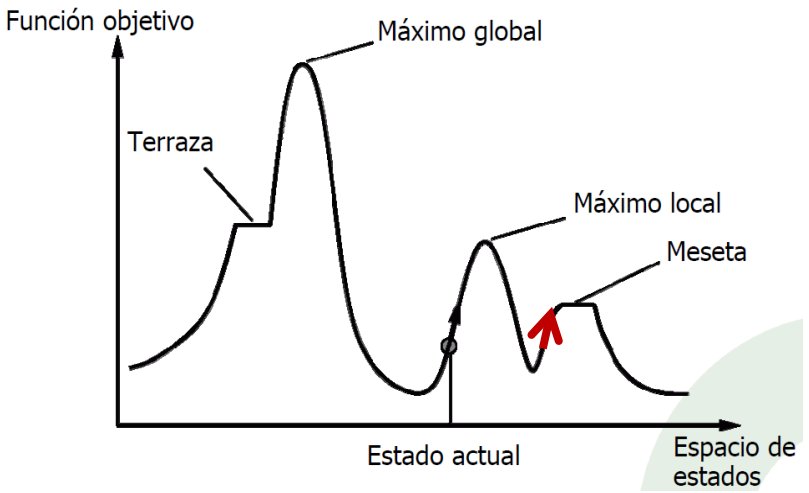
    Debe garantizar: factibilidad, baja cardinalidad |N(s)|, oportunidad de mejora, facilidad de evaluación, etc.

Ventajas:

- Requiere poca memoria
- A menudo encuentran soluciones razonables, en espacios de estados en los que la búsqueda global es inabordable

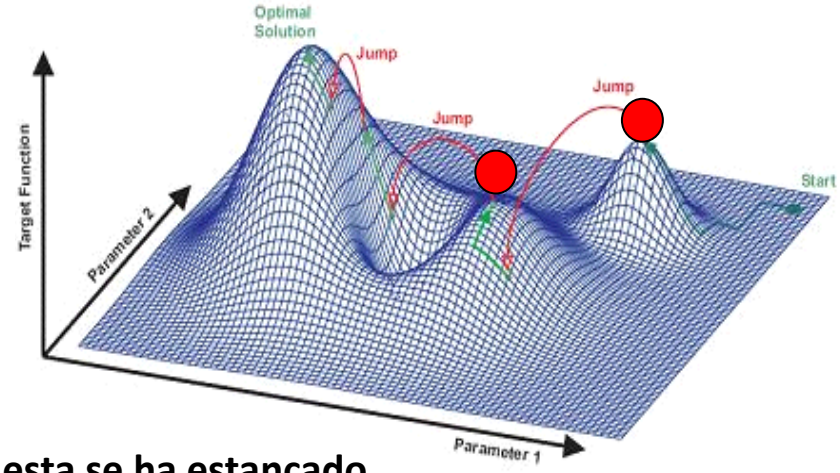
Inconvenientes:

- La búsqueda puede acabar en óptimo local
- Se obtienen soluciones localmente óptimas, pero pueden estar lejos del óptimo global
- Bucles (debido a no almacenar en memoria)



Alternativas para escapar de óptimos locales

- Salto a otra zona de búsqueda previa (backjumping)
- Avanzo al siguiente nivel, aunque sea peor
- Re-arranque (arranque múltiple)



Re-arranque *("si no sale a la primera, inténtalo otra vez")*

- **Se reinicia la búsqueda con una nueva solución cuando esta se ha estancado**
  Se repite varias veces, partiendo cada vez de un estado inicial distinto
- Si la probabilidad de éxito de una búsqueda individual es p, entonces el número esperado de reinicios es 1/p. Se queda con la mejor solución de todos los multiarranques.

No es propiamente una metaheurística

Ventaja: Escapa de valles

Desventaja: Perdida de la información de búsqueda en cada reinicio

¿Alternativas de reinicio?

Algoritmo Escalada con Reinicio (maximización)
$S_{actual} \leftarrow S_0$ *(*Solución_Inicial*),* $S_{mejor} \leftarrow S_{actual}$

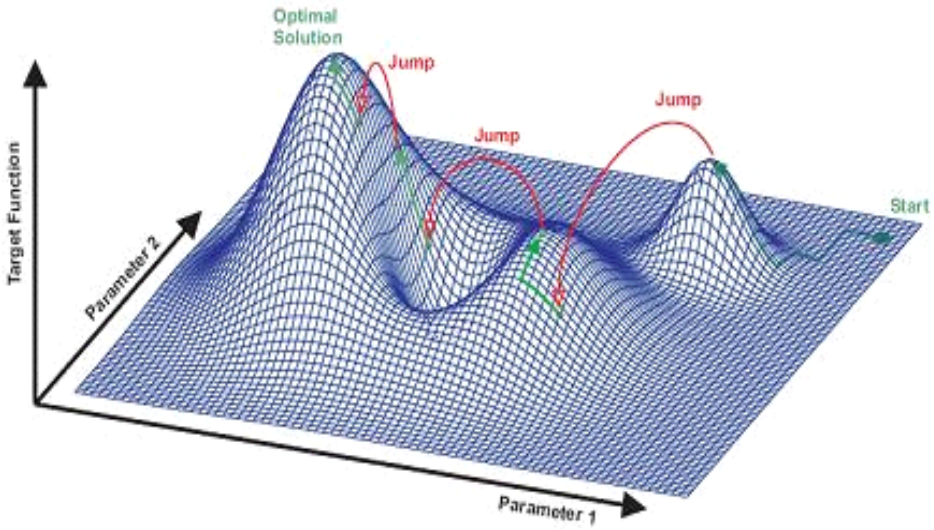
mientras no_condición_parada **hacer**
  $S_{nueva} \leftarrow$ Mejor $\{S \in$ Vecinos$(S_{actual})\}$
  si $f(S_{nueva}) > f(S_{actual})$
    entonces $S_{mejor} \leftarrow S_{nueva}$
      $S_{actual} \leftarrow S_{nueva}$
    sino $S_{actual} \leftarrow$ *Generar (*S_inicial)
  fin
Fin ; devuelve S_{mejor}

Alternativas de reinicio

a) Aceptar (a veces) movimientos que empeoran la calidad de la solución actual

b) Modificar la vecindad N(s) durante la búsqueda

c) Ampliar la búsqueda local

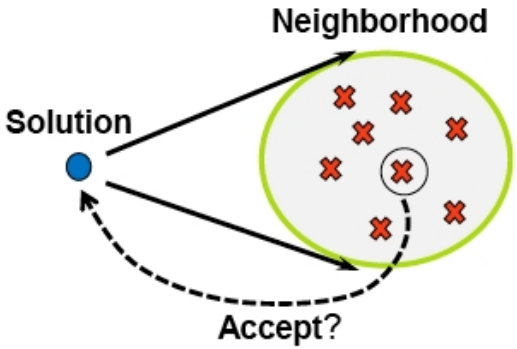


Metaheurística: estrategia de búsqueda global para **escapar de óptimos locales** de baja calidad

a) Metaheurísticas de **búsqueda no monótona**:

Permiten movimientos que no sean de mejora

Enfriamiento Simulado *(Simulated Annealing)*



b) Metaheurísticas de **entorno (vecindad) variable:**

Modifican la estructura de entornos locales que se aplica

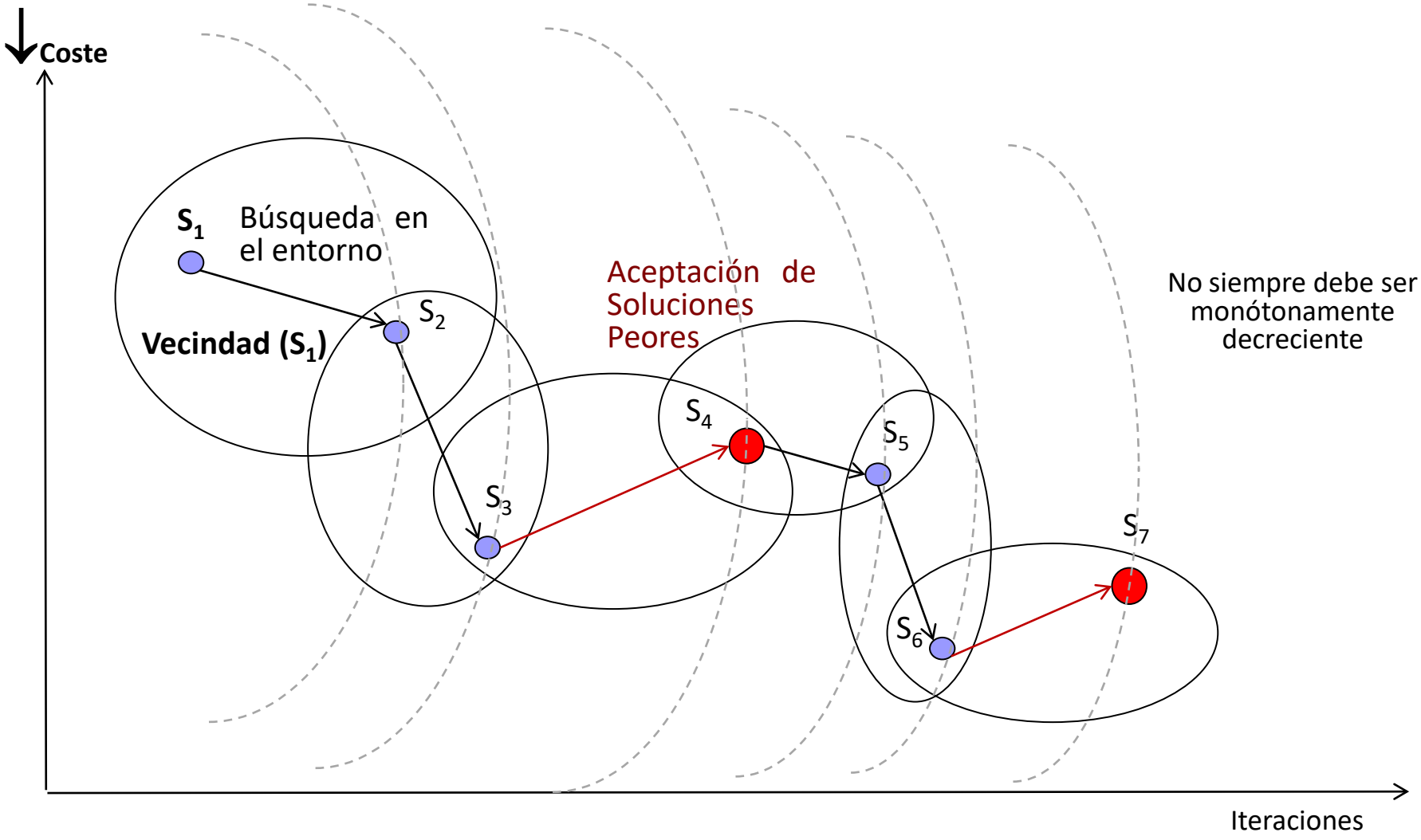
Búsqueda Tabú (búsqueda con memoria) y Vecindades Variables

c) Metaheurísticas de **búsqueda en haz**

Metaheurísticas de mejora

Búsqueda en un Espacio de Soluciones / Búsqueda Local

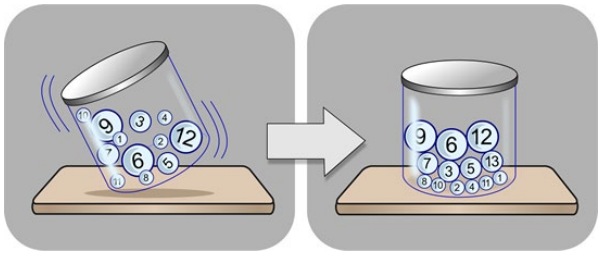
Metaheurísticas de Trayectoria / Búsqueda en entornos



Enfriamiento simulado *(Simulated Annealing, Recocido/Temple Simulado)*

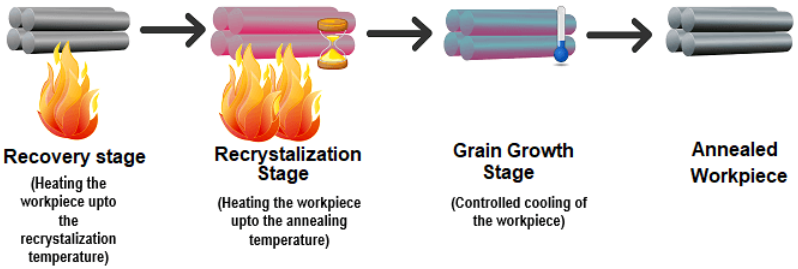
Kirkpatrick, Gelatt, Vecchi "Optimization by Simulated Annealing". Science 220 (4598):671–680

¿Cómo forjamos una espada?



SA. Ideas generales

Annealing: "Proceso de enfriar lentamente un material en estado líquido hasta un estado cristalino de mínima energía. Si la temperatura se reduce de manera suficientemente lenta, el material obtendrá su estado de más baja energía (ordenación perfecta)"



Annealing is a heat treatment in which the metal is heated to a temperature above its recrystallisation temperature, kept at that temperature for some time for homogenization of temperature followed by very slow cooling to develop equilibrium structure in the metal or alloy.

Consiste en:

1. Realizar una exploración **más amplia al principio**, ya que la solución final es relativamente insensible a los estados iniciales y se disminuye la probabilidad de caer en mínimos locales, valle o cresta en zonas iniciales de la búsqueda
2. **Admitir**, con cierta probabilidad, estados sucesores en los que f(x) empeore, a fin de poder salir de óptimos locales. Esta probabilidad disminuye conforme avanza la búsqueda
3. Concentrar (guiar) más la búsqueda en las **etapas finales**

Aplicaciones

SA desde 2022

SA. Esquema

Elección aleatoria de un vecino/sucesor del estado actual ***(≅Escalada estocástica)***

a) Si es **mejor,** se elige incondicionalmente

b) Si es **peor,** se acepta el estado con una probabilidad que depende de la temperatura (T) y el incremento de energía (ΔE), asimilado al decremento del valor de f(n)

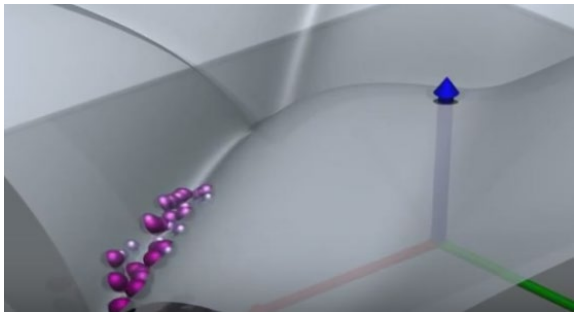
La temperatura (T) va bajando conforme avanza la búsqueda (a cada iteración)

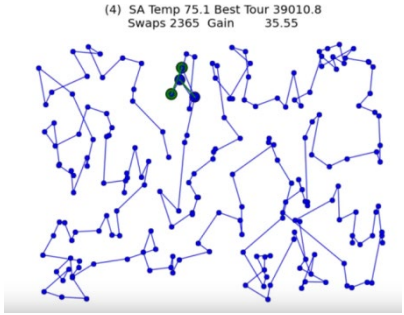
Por ello, la **probabilidad** de aceptar un estado que empeore el actual:

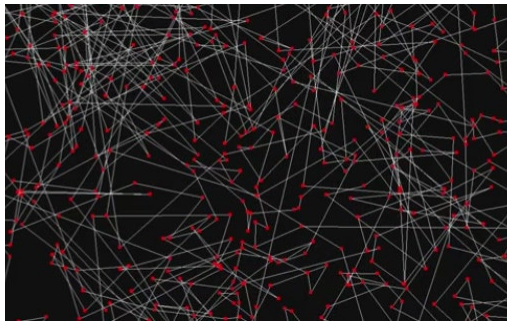
va bajando a medida que avanza la búsqueda…

…además de cuánto empeora la función objetivo

Hacia el final de la búsqueda, solo se admiten soluciones que mejoren o igualen la actual







SA. Ejemplo (maximización)

En cada iteración, en vez de elegir el mejor movimiento sucesor, se elige un sucesor/vecino **aleatorio**:

a) Si el movimiento mejora la situación (Δf>0), entonces es aceptado (Δf = f(vecino) - f(actual))

b) Si no mejora la situación (Δf≤0), se acepta el movimiento con una probabilidad menor a 1:

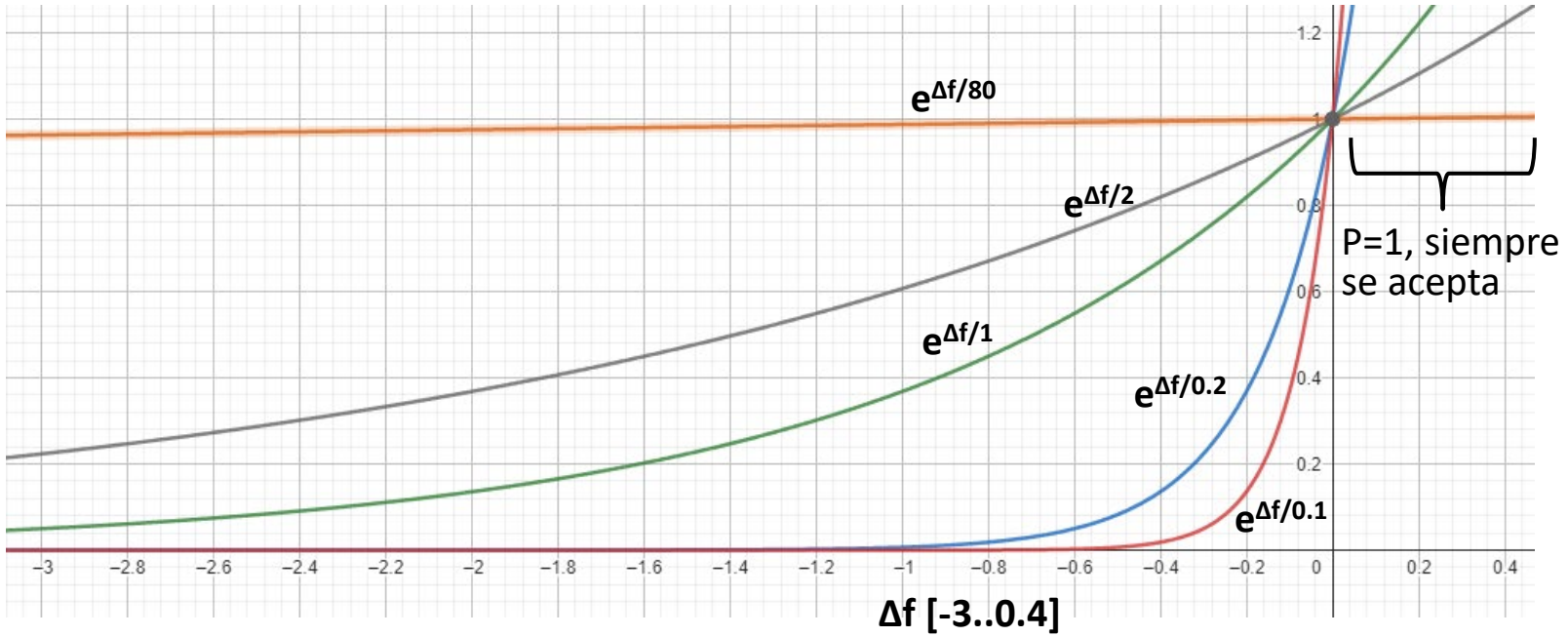
En general (a+b), **Probabilidad de aceptar al vecino = mínimo(1, $e^{\Delta f/T}$)**

Esta probabilidad se decrementa:

- Con el empeoramiento de la evaluación Δf (ya que Δf<0)
- Con la temperatura (T), que va bajando conforme avanza la búsqueda

Si minimización:

Δf = f(actual) - f(vecino)
o usar $e^{-\Delta f/T}$



SA. Ejemplo (maximización)

¿Cómo podemos decrementar la temperatura (T) conforme avanzan las iteraciones (i)?

T_{i+1} se calcula en base a una función que puede depender de múltiples factores; ej. $T_{INICIAL}$, iteración anterior *i*, temperatura anterior T_i, coeficiente *k*, etc.

Existen varias alternativas para definir:

a) $T_{i+1} = T_{INICIAL} - i*k$ (lineal)

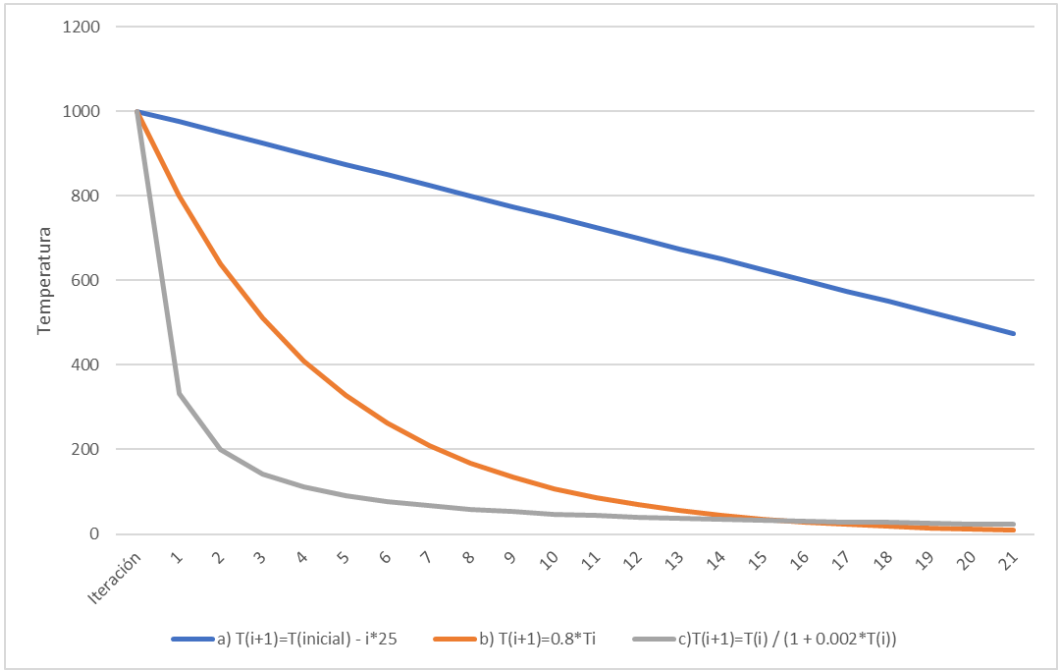
b) $T_{i+1} = k*T_i$, con $0<k<1$, típicamente {0.8, 0.99}

c) $T_{i+1} = T_i / (1 + k*T_i)$, $0<k<1$ pero con valor muy pequeño

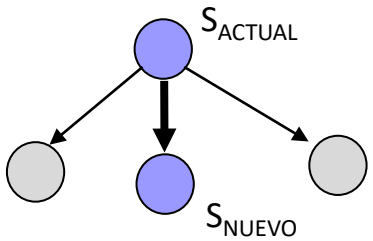
d) escalonada o por etapas: T se mantiene durante *k* iteraciones y luego se decrementa, se mantiene durante otras *k* iteraciones y se vuelve a decrementar...

Si la variación de T es lenta y $T_{INICIAL}$ es alto, la probabilidad de llegar a una buena solución es mayor, pero la búsqueda es más costosa

El ajuste de los parámetros $T_{INICIAL}$ y *k* dependen del problema. **Ajuste experimental**



SA. Algoritmo (maximización)



$S_{ACTUAL} = S_{INICIAL}$; $S_{MEJOR} = S_{ACTUAL}$; i=0; $T_i = T_{INICIAL}$; ***;Parámetros iniciales***

Mientras *"Existan sucesores de S_{ACTUAL}"* y *"criterio_terminación=falso"* **hacer**

$S_{NUEVO} \leftarrow$ Operador-aleatorio (S_{ACTUAL}) ***; Movimiento aleatorio ¡no necesariamente el mejor!***

$\Delta f \leftarrow f(S_{NUEVO}) - f(S_{ACTUAL})$ ***; ¿Mejora o Empeora?***

Si $\Delta f > 0$ entonces ***; Mejora la solución: se acepta el movimiento***

$S_{ACTUAL} \leftarrow S_{NUEVO}$

Si S_{NUEVO} mejor que S_{MEJOR} entonces $S_{MEJOR} = S_{NUEVO}$

Si_no ***; No mejora: el movimiento se puede aceptar o no***

$S_{ACTUAL} \leftarrow S_{NUEVO}$ con probabilidad $e^{\Delta f/T}$ ***; Podría no actualizarse S_{ACTUAL}***

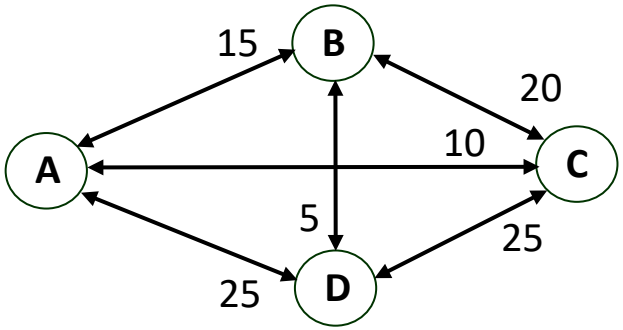
T_{i+1} = *valor de la función que define el decremento de la temperatura (ej. a, b, c, d…)*

i=i+1

finMientras

Devolver S_{MEJOR}

Ejemplo 1. Viajante de comercio (minimización)



$S_{INICIAL}$= (A, B, D, C, A); S_{ACTUAL}= $S_{INICIAL}$

S_{MEJOR} = S_{ACTUAL}; i=0; T_i=1000; k=0.01

Δf = f(actual) - f(vecino)

$T_{i+1} = T_i/(1 + k*T_i)$

Solución inicial
i=0, T_0=1000

(A, B, D, C, A) 55

Elección Operador aleatoria

(A, D, B, C, A) 60

Δf = 55-60=-5

Probabilidad de elección = $e^{\Delta f/T} = e^{-5/1000} = 0.995$

Se acepta. S_{ACTUAL}=(A, D, B, C, A)

T_1=1000/(1+0.01*1000) = 90.91, i=1

Elección Operador aleatoria

85 (A, B, C, D, A)

(A, C, B, D, A) 60

Δf = 60-85=-25

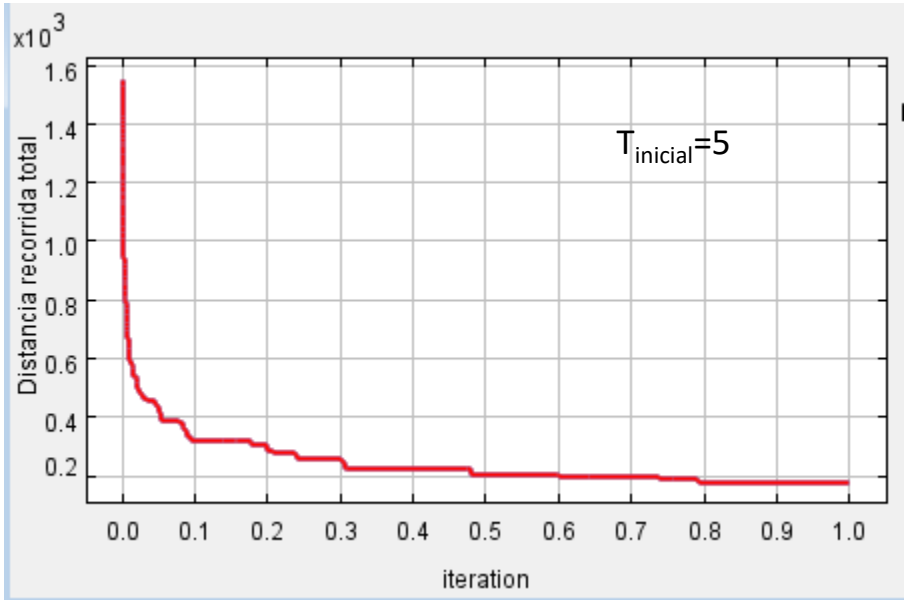
Probabilidad de elección= $e^{\Delta f/T} = e^{-25/90.91} = 0.76$

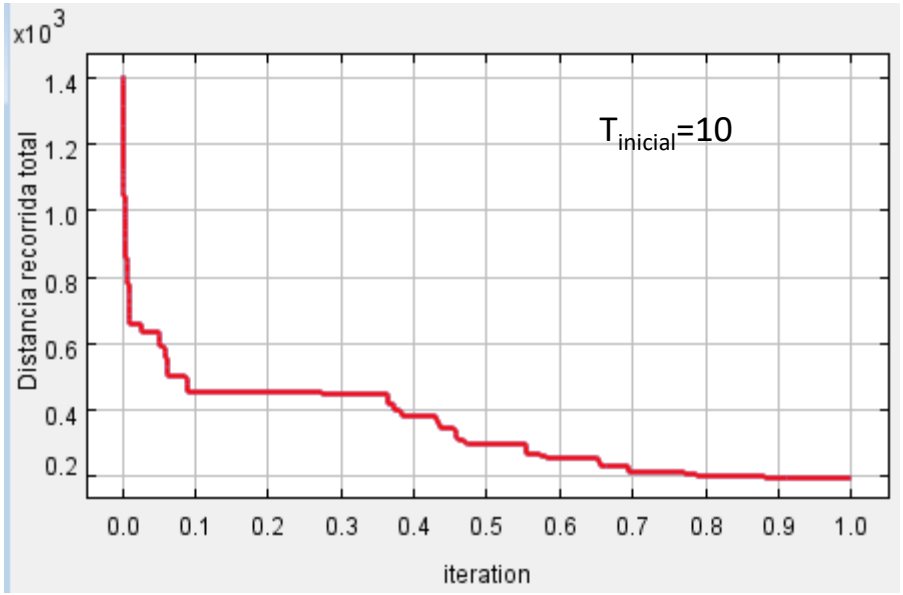
No se acepta.

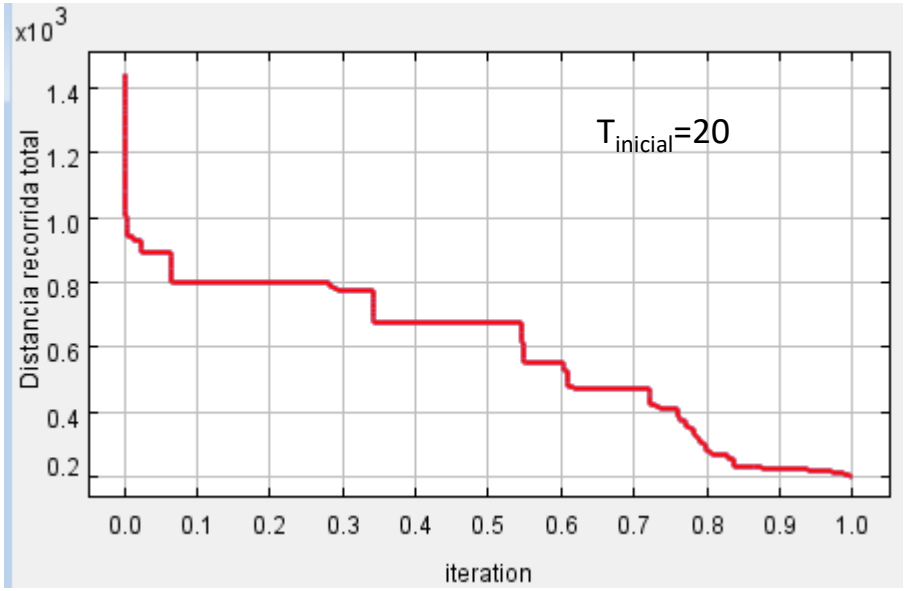
T_2=90.91/(1+0.01*90.91) = 47.62, i=2

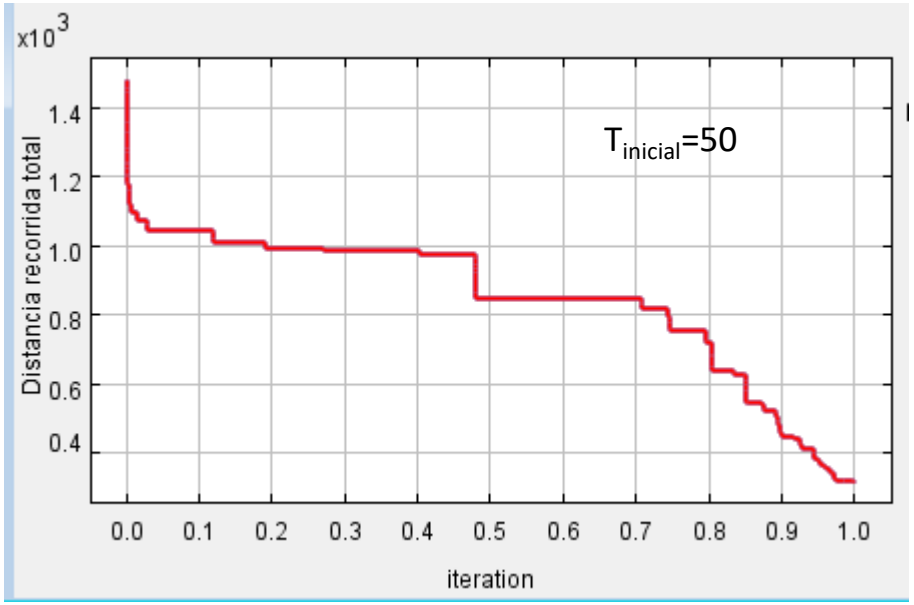
Ejemplos de convergencia *vs.* $T_{inicial}$

Iteraciones x 10^5. $T_{i+1} = T_{inicial} - i*k$



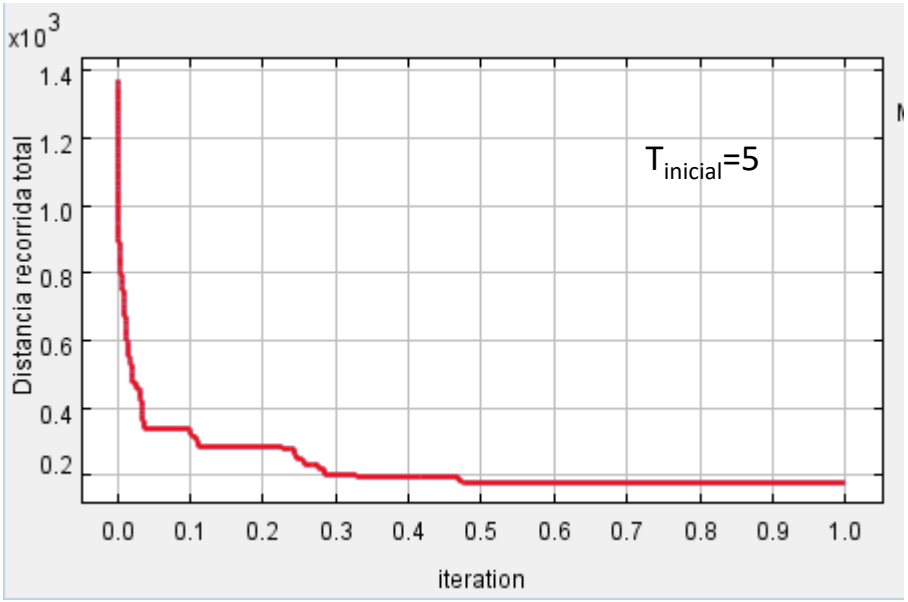


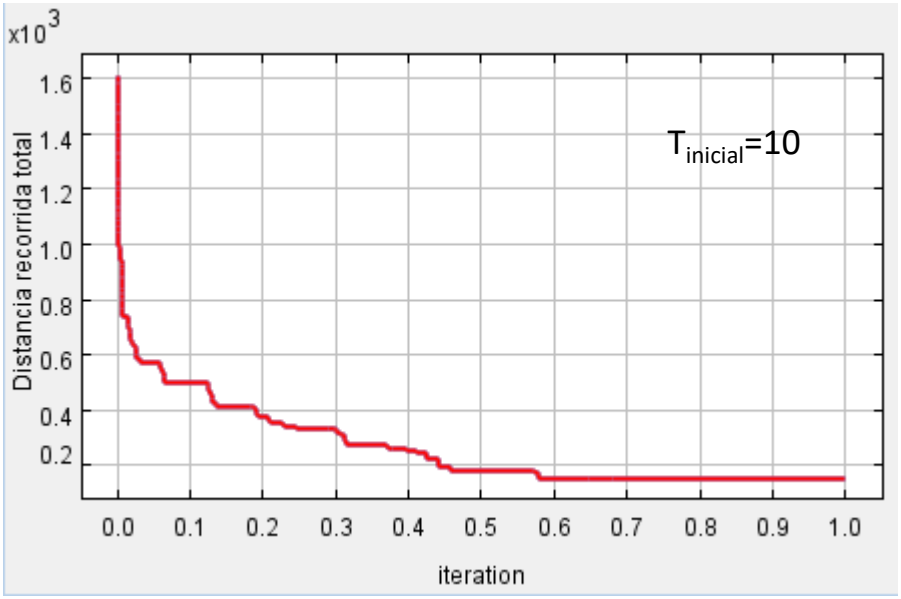


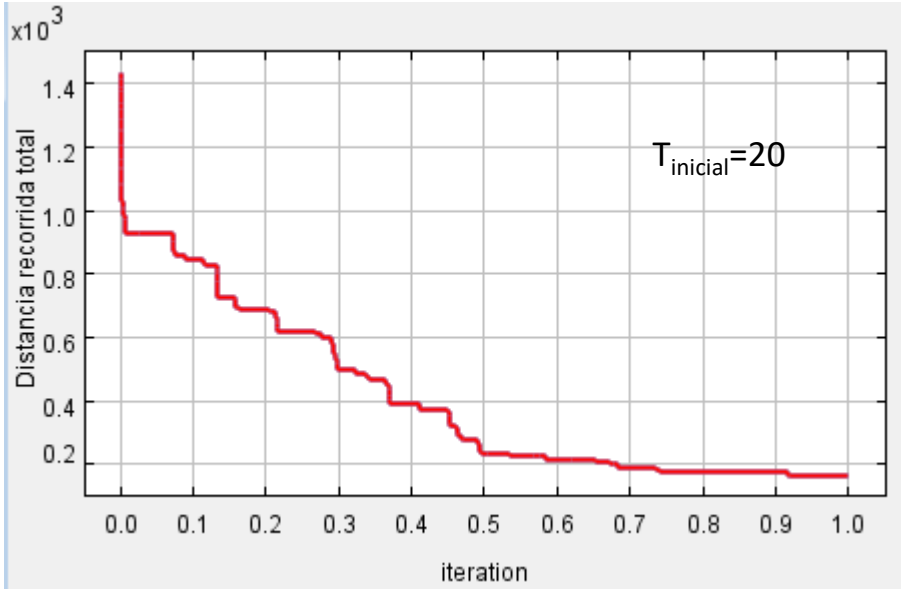


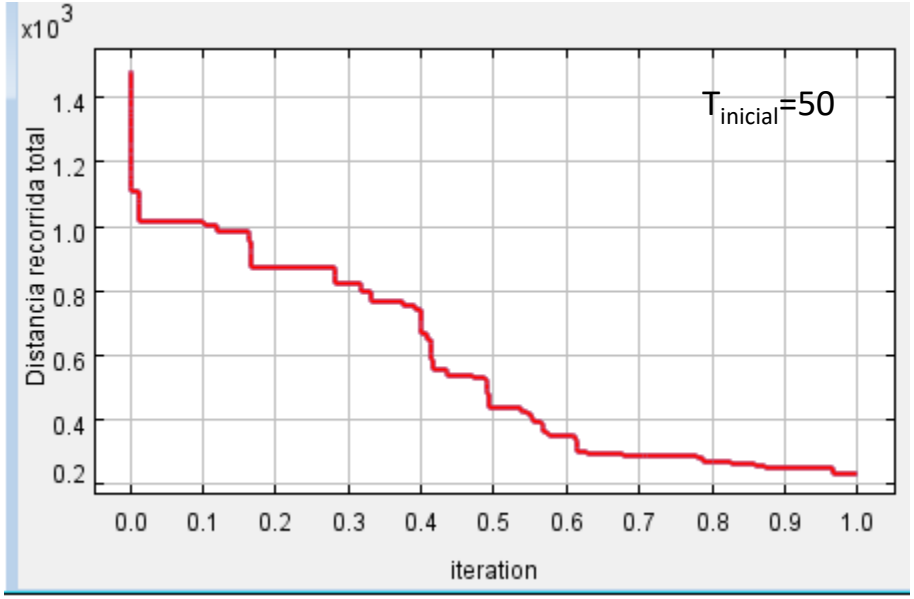
Ejemplos de convergencia *vs.* $T_{inicial}$

Iteraciones x 10^5. $T_{i+1} = k*T_i$



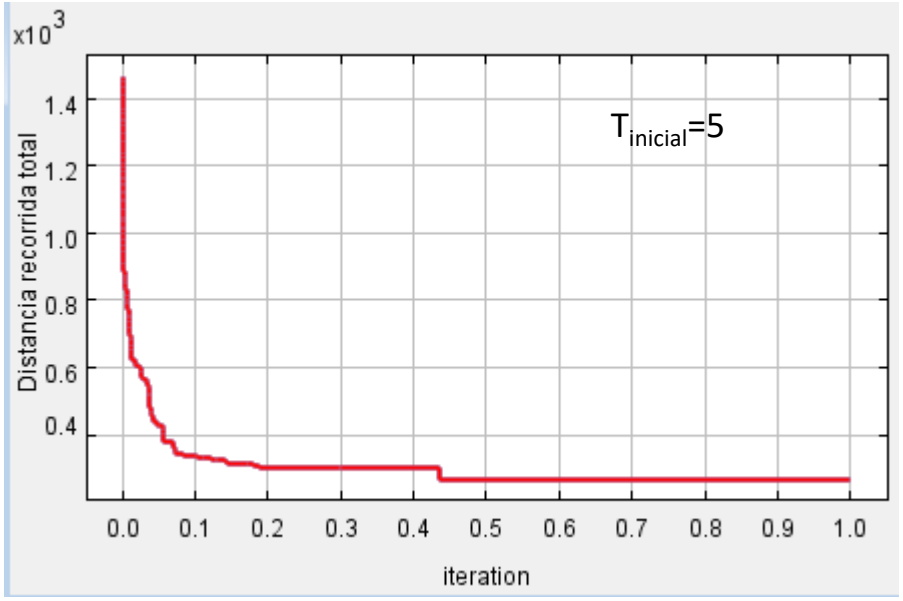


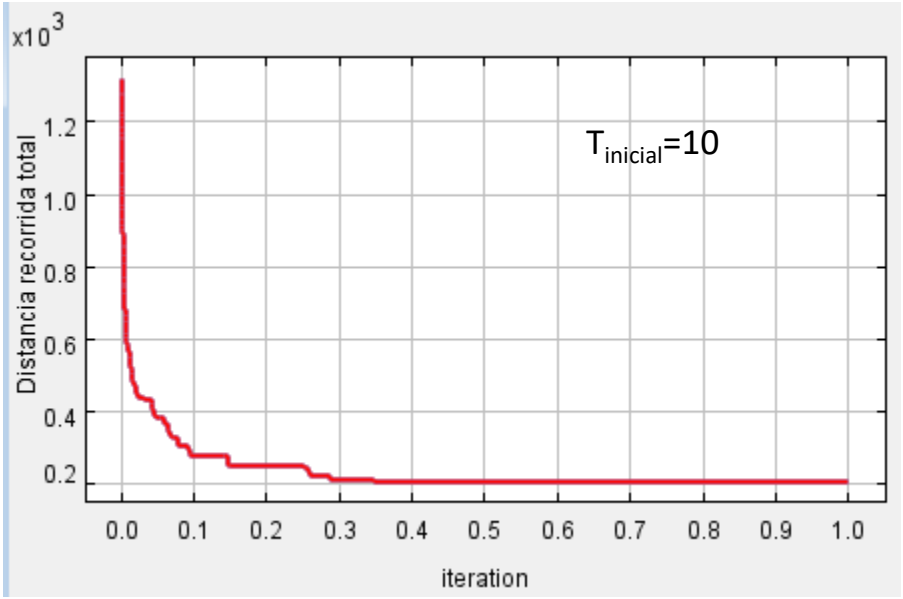


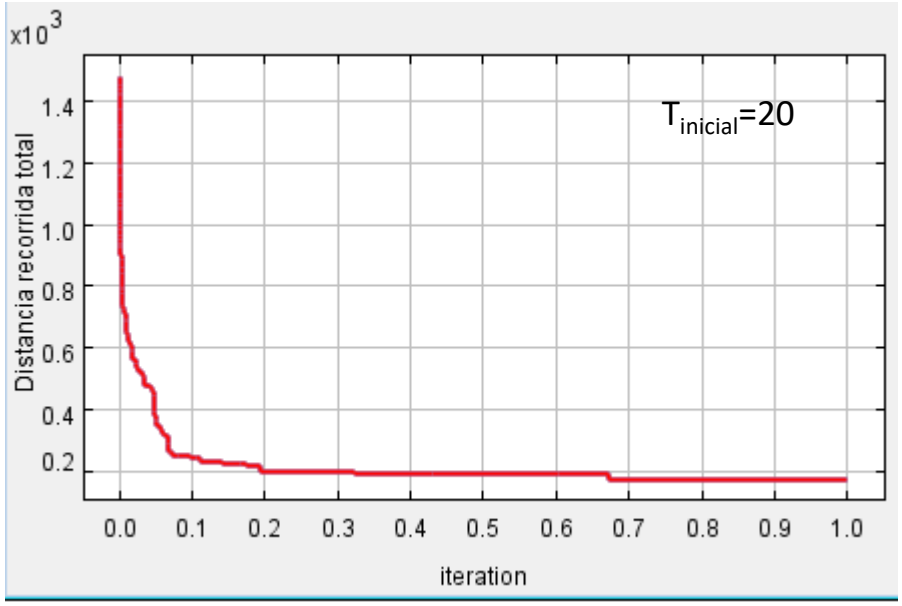


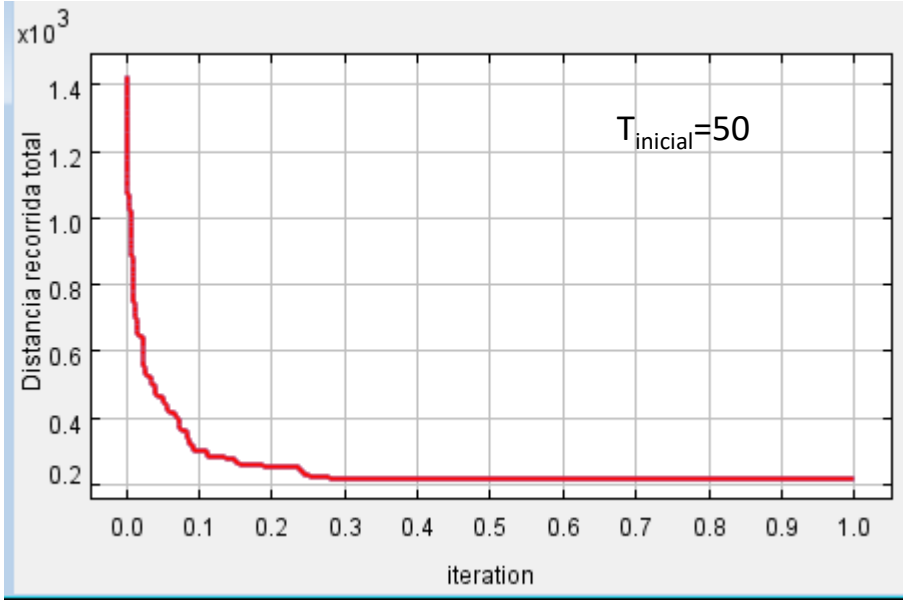
Ejemplos de convergencia *vs.* $T_{inicial}$

Iteraciones x 10^5. $T_{i+1} = T_i / (1 + k*T_i)$

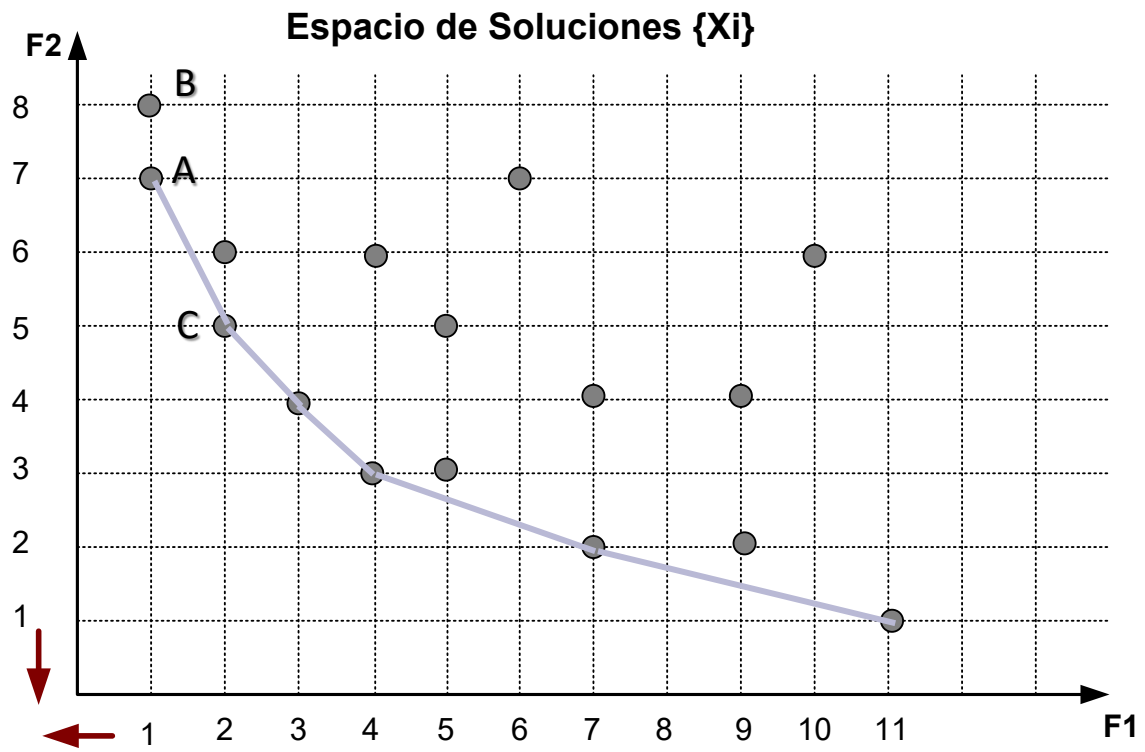








Optimización multi-objetivo. Frontera de Pareto (minimización de F1 y F2)



Mecanismo equivalente al de metaheurísticas evolutivas

Se ordenan las soluciones según las funciones fitness objetivo (F1, F2, ….)

	F1	F2
A	1	7
B	1	8
C	2	5
	2	6
	3	4
	4	3
	4	6
	5	3
	5	5
	6	7
	7	2
	7	4
	9	2
	9	4
	10	6
	11	1

Búsqueda en Vecindades Variables (Variable Neighborhood Search - VNS). *Hansen, Mladenovic. 1997*

Modificación de la búsqueda local que utiliza distintas vecindades a lo largo de la búsqueda

Estructura de vecindad: para cada $s \in S$, se define el conjunto $N(s) \subseteq S$ de soluciones "vecinas" de s

N(s) es el conjunto de soluciones a las que se puede llegar desde s aplicando el procedimiento N

El criterio para definir las soluciones vecinas de cada s, N(s), es muy amplio y con alternativas

N(s) no tienen por qué ser las más cercanas a s (soluciones-vecinas ≠ cercanas)

Problema: Óptimo Local

- Un óptimo local en una estructura de vecindad, $N_i(s)$, no es necesariamente un óptimo local con otra estructura de vecindad, $N_j(s)$
- Un óptimo global es óptimo local con todas las posibles estructuras de vecindades

Idea:

$\Rightarrow$ Definir diversas estructuras de vecindad N_1, N_2, N_3, ….N_k

Similar a realizar un salto 'global' a otra zona (exploración)

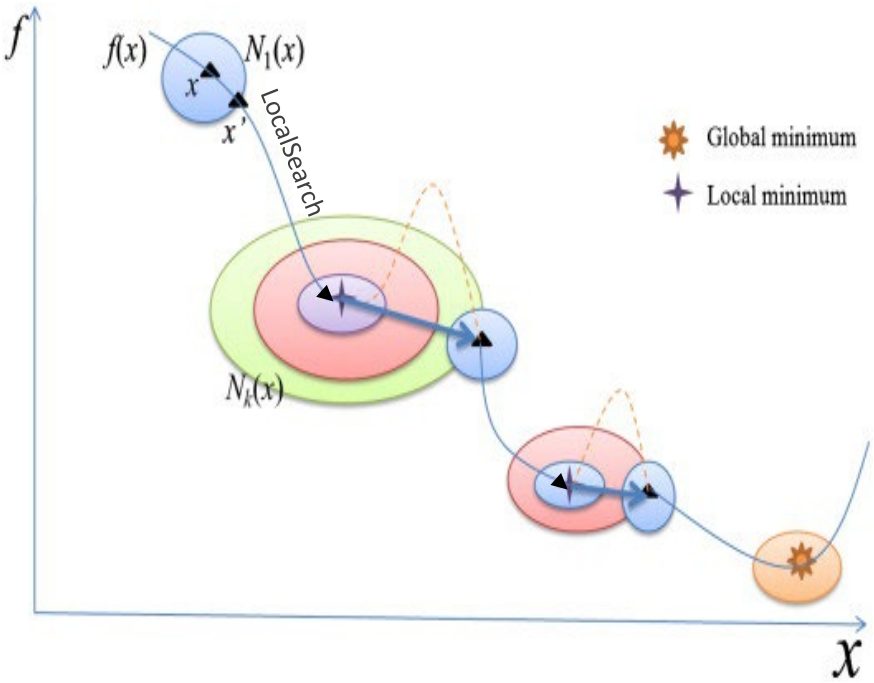
Esquema Búsqueda Local: N(s) es fijo

s ← Solucion_Inicial

Repeat

s ← Mejorar(s, N(s))

Until no hay mejora posible



Se define un conjunto de estructuras de vecindad: $\{N_k\}$, $k = 1,\ldots\ldots,k_{max}$

s ← SolucionInicial ()

Mientras no_fin *;recorre la estructura de vecindad*

k ← 1

Mientras $k < k_{max}$

s´ ← SeleccionarRandom (N_k(s))

s´´ ← LocalSearch(s´) *;hasta máx-local*

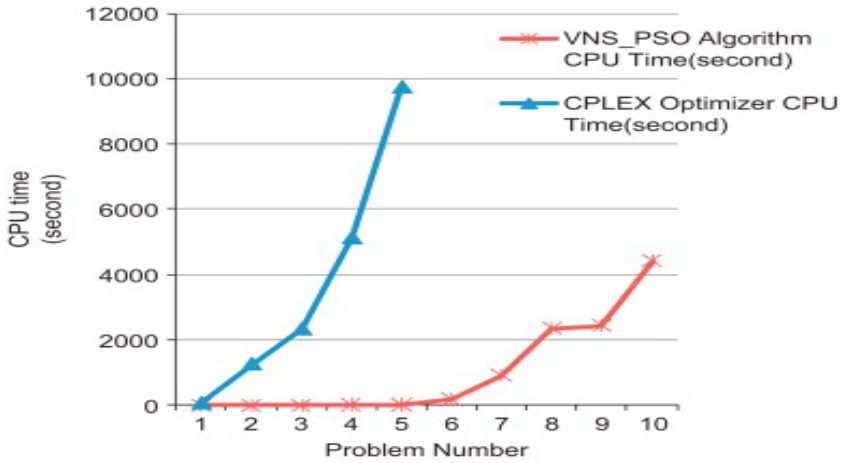
if f(s´´) < f(s) *then* s ← s´´, k ← 1

else k ← k+1

end if

End_mientras

End_mientras



A hybrid approach based on the variable neighborhood search and particle swarm optimization for parallel machine scheduling problems—A case study for solar cell industry Int. J.of Production Economics V.141, I.1, 2013

Recomendaciones para diseñar un algoritmo de ES. Conclusiones

- **Indicado:**

  En problemas grandes en los que el óptimo está rodeado de muchos óptimos locales

  Cuando una elección aleatoria es tan buena como otra cualquiera

- **Selección de la solución de partida:**

  Puede construirse de forma elaborada (ej. resultado de un AG)

- **Es muy importante parametrizar el problema:**

  Elección de $T_{inicial}$, cómo se decrementa la temperatura T_{i+1} (no debería ser muy rápidamente para obtener mejores soluciones), probabilidad de aceptar un vecino que empeora, etc.

- **Es muy importante definir los vecinos (fijos *vs.* variables):**

  Puede ayudar a escapar de óptimos locales (exploración). Si una solución s tiene un vecino s', entonces s' debería tener como vecino también a s (enfoque reversible)

  ¿Se generan pocos vecinos de forma inteligente o muchos de forma aleatoria?

- **Condición de terminación:**

  Lo más habitual es que la condición de terminación sea la convergencia del algoritmo, número prefijado de iteraciones, alcanzar una determinada temperatura o tiempo disponible

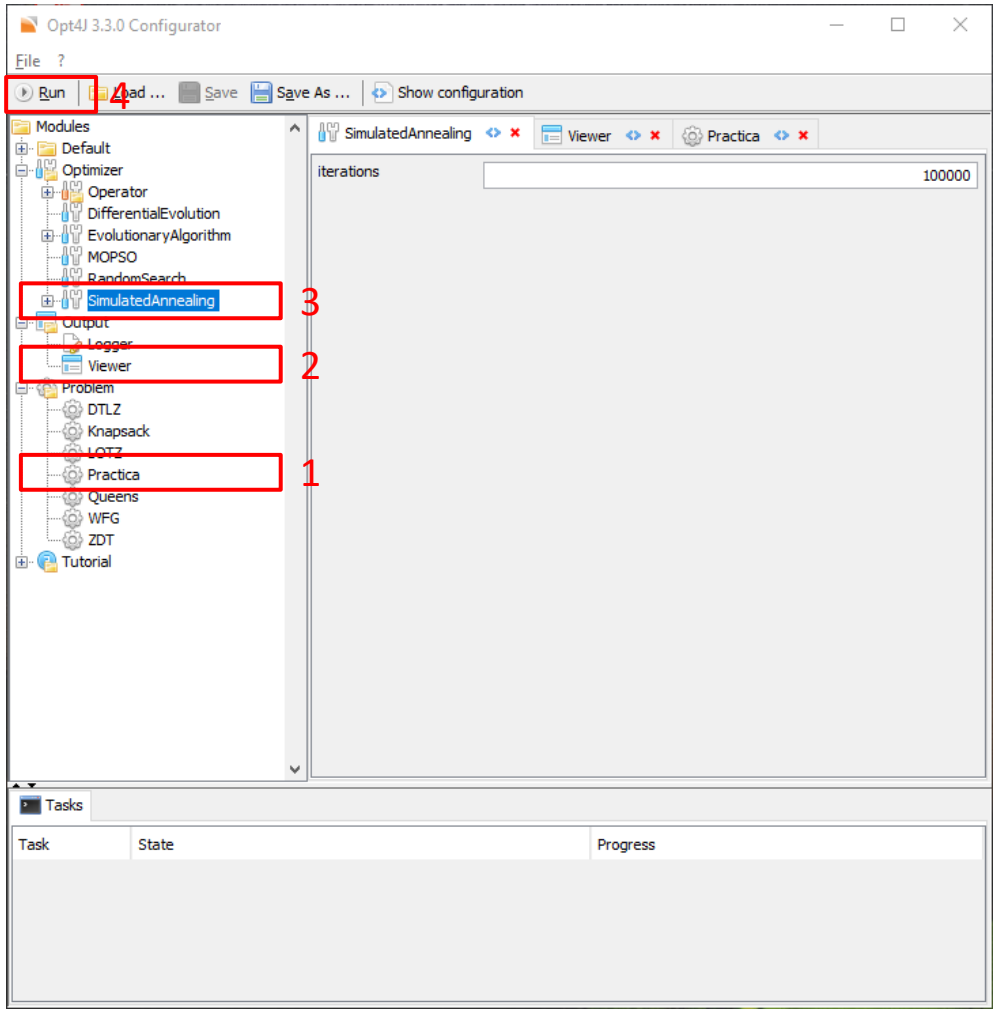
  ¿Se permite un enfriamiento y posterior recalentado **(enfriamiento no monótono)**?

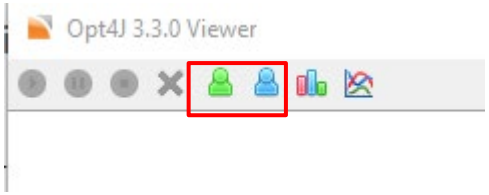
- **Función de Aptitud:**

  Debe hacer una presión adecuada y reflejar la calidad de la solución en el problema

Ejercicio en Poliformat. Adivinar frases

1. Descargar “Tema 2 EjercicioAG.jar” de Poliformat
2. En **estricto orden 1-2-3**, elegir Practica, Viewer y SimulatedAnnealing
3. Seleccionar el valor de los parámetros (iterations)
4. Ejecutar Run
5. Ver los mejores individuos y tratar de adivinar la frase y el autor. Para ello probad con otros parámetros





"The general employee scheduling problem: an integration of MS and AI". Glover & McMillan. Computers and Op. Research

Introduce una memoria (lista tabú) para recordar y evitar los problemas básicos de la búsqueda local:

- **evitar parar en mínimos locales:**

  continúa con la búsqueda local, aunque sea con movimientos que no mejoren la solución, para evitar caer en valles

- **evitar entrar en ciclos:**

  usa la memoria temporal (lista tabú), que guarda la historia reciente de la búsqueda, para evitar estados repetidos

Conceptos:

- Se parte de una solución, o estado inicial ($x \in X$), que se intenta mejorar
- Tamaño de la lista tabú es **limitado. Suele ser una lista circular** (eliminando el elemento más viejo)
- Lista Tabú (T): movimientos o estados prohibidos. ***(Búsqueda de entorno variable)***
- En su versión más simple, la lista tabú es la lista de los recientes estados visitados, pero suele ser un conjunto de estados-tabú no deseados, movimientos o características no deseadas, etc.

Algoritmo Búsqueda Tabú (minimización)

Inicialización (estado inicial: Sol ∈{Soluciones})

Sol* = Sol	(mejor solución hasta el momento)
k = 0	(contador de iteración)
Tabu = ∅	(lista tabú de estados/movimientos prohibidos: *lista circular limitada*)
Suc(Sol)	(estructura vecindario o entorno frontera, estados sucesores, accesibles desde Sol)
f(s)	(fitness de la solución s)

Mientras no_fin hacer

La búsqueda no se detiene cuando empeora f(n)

Identificar Suc(Sol)

Si Suc(Sol) - Tabu = ∅ entonces parar

si no

k = k + 1

seleccionar Sol , Sol = OPTIMO(Suc(Sol) - Tabu)

Si f(Sol) < f(Sol*) entonces Sol* = Sol

Guarda en Sol* la mejor solución

Actualizar 'Tabu' (añadir movimiento actual y eliminar elemento viejo, si procede)

Devolver Sol* **(mejor solución encontrada)**

La lista tabú T de estados prohibidos permite 'saltar' a estados no óptimos y prohíbe volver a estados previos (o prohibidos)

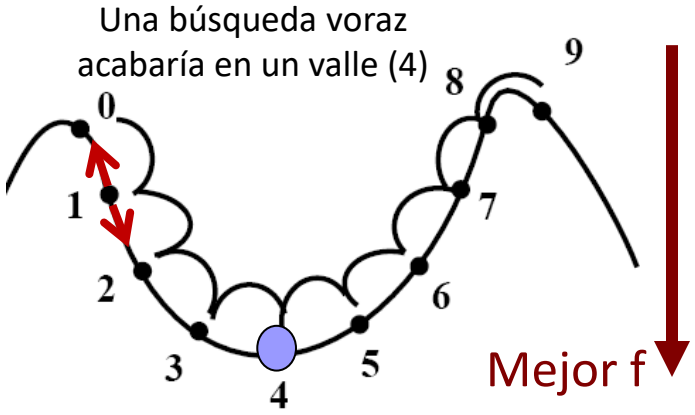
Ejemplo 1 (salir de valles y óptimos locales - minimización)

- **Suposición**: cada estado solo tiene dos sucesores y la lista tabú es de tamaño 3 (inicialmente vacía)
- Inicialmente estamos en 1

  Se pueden hacer dos movimientos (0 y 2)

  Se elige el mejor (2) y se actualiza la lista tabú: registramos el movimiento inverso T={mov(2; 1)} para evitarlo en el futuro
- Desde 2 movemos al estado 3 (el mejor de los permitidos dada la lista tabú actual). Actualizamos la lista tabú
- Desde 3 movemos al 4. Actualizamos la lista tabú, que ahora tiene tres elementos: T={mov(4; 3); mov(3; 2); mov(2; 1)}
- Desde 4 movemos al punto 5, peor según su fitness, pero es el mejor (el único) de los permitidos por la lista tabú: moverse a 3 no está permitido

  Actualizamos la lista tabú, que ya ha llegado a su límite: se elimina el elemento más viejo: T={mov(5; 4); mov(4; 3); mov(3; 2)}
- …
- Estando en 8, nos movemos al punto 9, y salimos del mínimo local con T={mov(9; 8); mov(8; 7); mov(7; 6)}



La búsqueda no se detiene cuando empeora f(n):

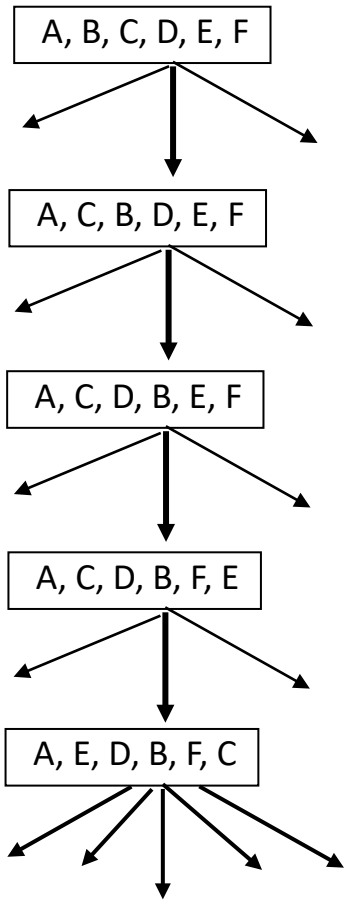
- Normalmente la lista tabú se implementa como una lista circular, eliminando los primeros añadidos
- Típicamente, se va rellenando con los movimientos inversos a los realizados (o los estados ya visitados para evitar ciclos)

Ejemplo 2. Problema del viajante de comercio

Ciudades: A, B, C, D, E, F

Solución Inicial: A, B, C, D, E, F

La lista tabú representa **movimientos** prohibidos



Lista Tabú = {}

Búsqueda:

Obtención de un mejor estado sucesor

Sol = OPTIMO(Suc(Sol) - Tabu)

Permuto (B C)

Lista Tabú = {(B C)}

Permuto (B D)

Lista Tabú = {(B D), (B C)}

Permuto (F E)

Lista Tabú = {(F E), (B D), (B C)}

Permuto (E C)

Lista Tabú = {(E C), (F E), (B D), (B C)}

El **tamaño** de la lista tabú es acotado y es un parámetro importante a definir

En este estado, no puedo realizar las permutaciones de la lista tabú, es decir, están prohibidos los estados sucesores:

A, C, D, B, F, E

A, F, D, B, E, C

A, E, B, D, F, C

Búsqueda Tabú. Conclusiones

- **Es muy importante parametrizar el problema. El tamaño de lista Tabú es esencial**

  Controla el proceso de búsqueda, combinando **explotación** (búsqueda local) con **exploración** (búsqueda global, por elementos menos óptimos no presentes en la lista tabú)

- **Balance exploración/explotación:**

  Tamaño memoria pequeño (a corto plazo). Los elementos se mantienen poco tiempo en la lista tabú, evitando repeticiones en la región en curso – más intensificación

  Tamaño memoria mediano (a medio plazo), para investigar alguna región potencialmente buena con más detalle

  Tamaño memoria grande (a largo plazo). Los elementos se mantienen más tiempo en la lista tabú, bloqueando todas las zonas ya visitadas y explorando nuevas alternativas – más diversificación

  El tamaño de esta lista puede ser variable durante el proceso de búsqueda

- **Se puede plantear un criterio de aspiración**

  Se permiten movimientos prohibidos por la lista tabú bajo determinadas condiciones. Ej: si se mejora la función objetivo, si la solución tiene una característica deseada, etc.

  Si se produce una aspiración, se reinicia la lista tabú

Búsqueda por Haz Local (Beam Search)

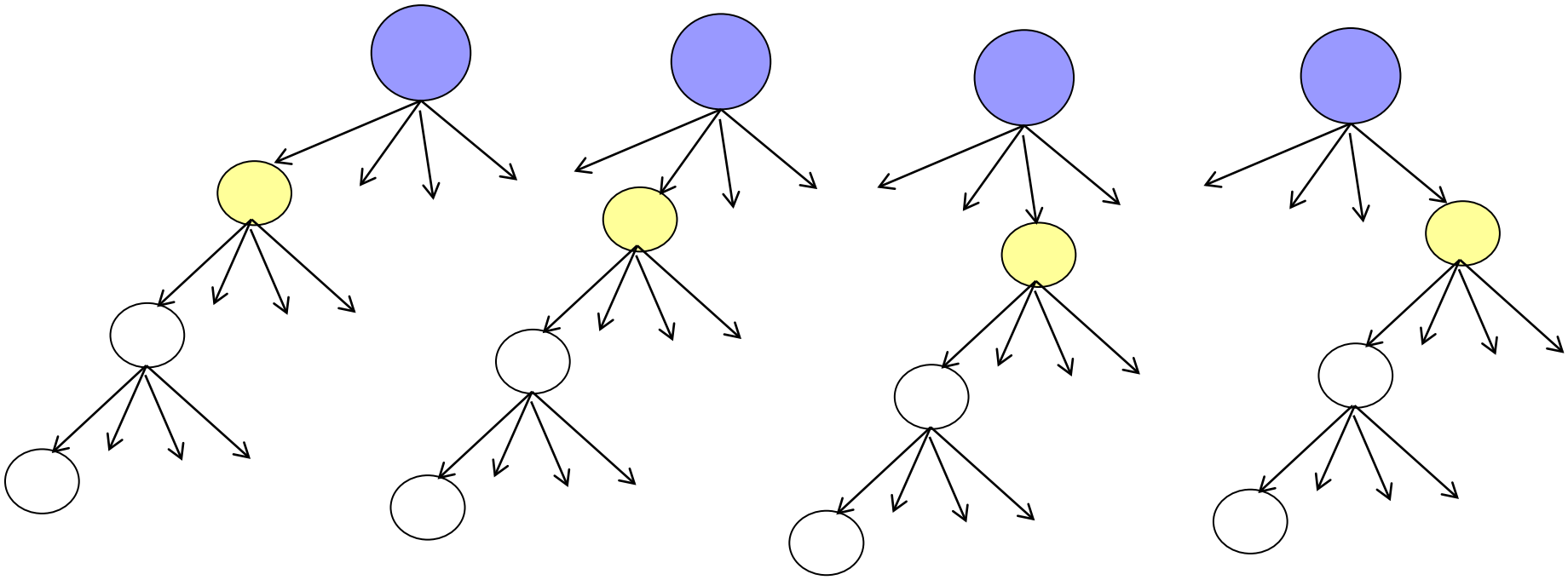
"Speech Understanding Systems: A Summary of Results of the Five-Year Research Effort. Dep. Computer Science.", Reddy, D. Raj, Carnegie Mellon University

Idea: escalada con re-arranque (multiarranque, Random restarting, backjumping…)

Se reinicia la búsqueda con una nueva solución cuando esta se ha estancado

Se repite *w* veces la búsqueda, partiendo cada vez de un estado inicial distinto, generado aleatoriamente: "*si no te sale a la primera, inténtalo otra vez*"

Requiere más cálculo, pero suele encontrar mejores soluciones

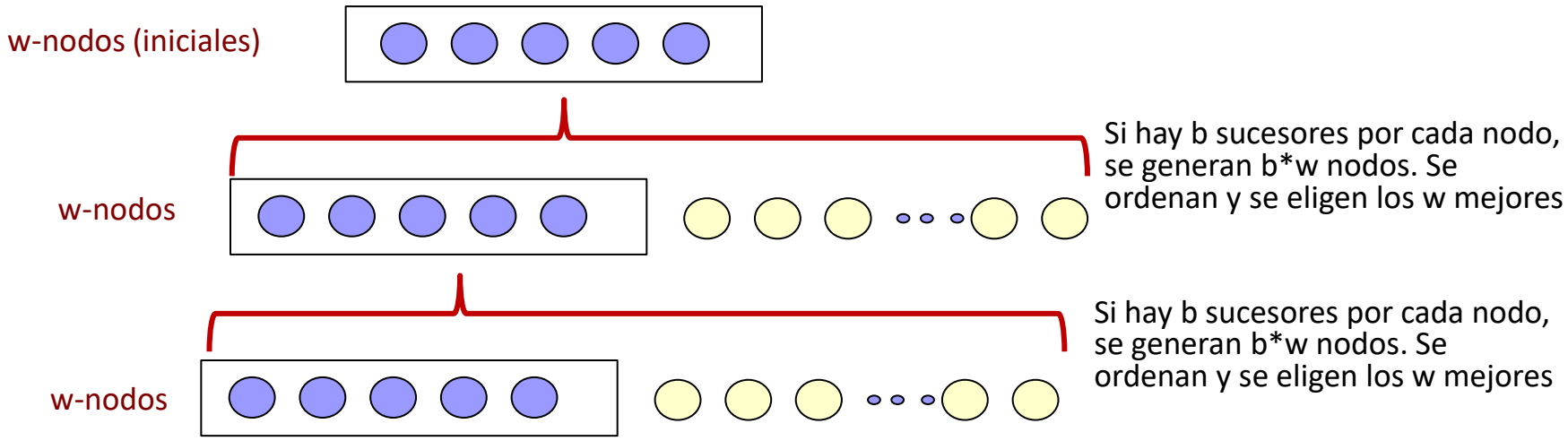


Búsqueda por Haz Local (Beam Search)

Es una reducción de una Búsqueda Sistemática, limitando la memoria requerida (a costa de la completitud y admisibilidad):

Se mantiene en memoria y se expande una ventana limitada de *w* nodos (similar a búsqueda en anchura con esos w nodos)

- Comienza con w estados generados aleatoriamente. Si alguno es objetivo (o se cumple la condición de terminación), se detiene la búsqueda
- En cada iteración, se generan todos los sucesores de los w estados y se ordenan según f(n)
  - Si alguno es objetivo o se cumple condición de terminación, se detiene la búsqueda
  - Si no, se seleccionan los w mejores sucesores (haz) y se repite el proceso
- ¡Diferente a lanzar en paralelo w escaladas, que generarían w-hilos de búsqueda distintos! En Haz Local, si un hilo genera los mejores sucesores, los w hilos de búsqueda seguirán por ese camino
- La amplitud/ventana del haz (w) puede ser fija o variable



Algoritmo Búsqueda por Haz Local

```
Start
Take the inputs
NODE = Root_Node & Found = False
If : Node is the Goal Node,
     Then Found = True,
Else :
     Find SUCCs of NODE if any, with its estimated cost&
     store it in OPEN List
While (Found == false & not able to proceed further), do
{
     Sort OPEN List
     Select top W elements from OPEN list and put it in
     W_OPEN list and empty the OPEN list.
     for each NODE from W_OPEN list
     {
         if NODE = Goal,
             then FOUND = true
         else
             Find SUCCs of NODE. If any with its estimated
             cost & Store it in OPEN list
     }
}
If FOUND = True,
    then return Yes
else
    return No
Stop
```

criterio de parada

ventana limitada de *w* nodos

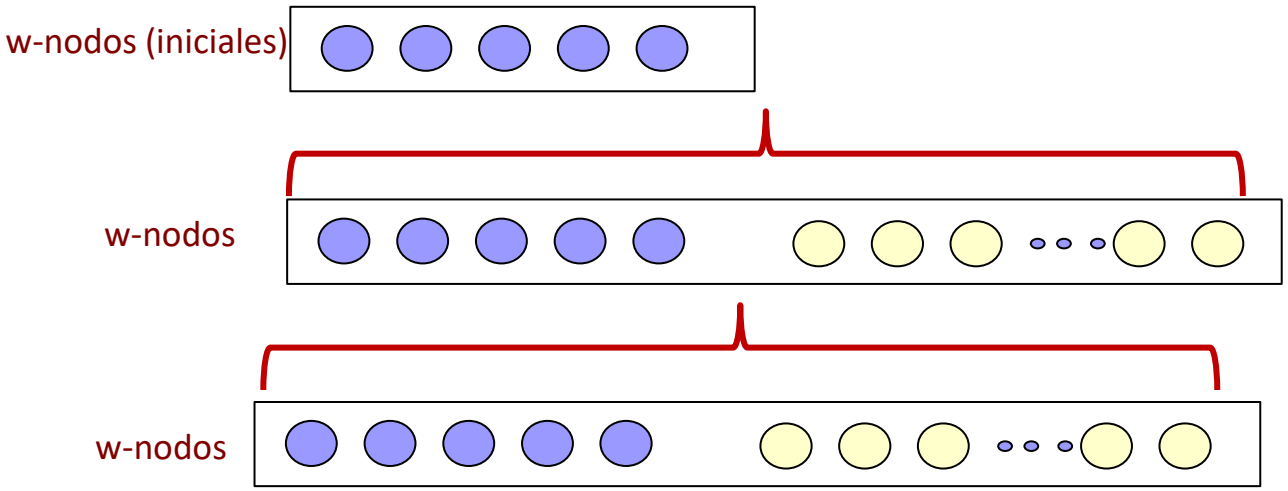
Variante. Búsqueda por haz estocástica

- La búsqueda por haz local puede concentrarse rápidamente en pequeñas zonas del espacio de estados. Sería interesante poder escapar de esa zona (exploración en otras zonas)
- La búsqueda por haz estocástica trata de combinar la explotación de las zonas mejores con la exploración de las zonas aparentemente peores

➢ En vez de elegir los w-mejores sucesores, se eligen w-sucesores con una probabilidad que es función de su fitness:

Los mejor valorados tienen mayor probabilidad de ser elegidos, aunque no siempre lo serán

- Guarda relación con metaheurísticas basadas en la selección natural (métodos poblacionales)



En todo momento, elección aleatoria de w nodos de todos los generados, proporcional a su f(n)

Búsqueda por Haz Local. Conclusiones

- **Es muy importante parametrizar el problema**

  El valor de w es esencial. Permite reducir el uso de memoria, a costa de perder completitud y optimalidad

  w=1 es equivalente a una búsqueda voraz. La búsqueda por Haz Local no necesariamente tiene que encontrar mejores soluciones que búsqueda voraz

  w>1 ofrecerá mejores soluciones, pero con una mayor complejidad espacial y temporal

  w=∞ sería equivalente a búsqueda en anchura

- **Se puede tratar de paralelizar en múltiples hilos, aunque compartiendo los w-mejores sucesores entre todos los hilos**

- **Balance exploración/explotación:**

  La búsqueda por haz estocástica incrementa la exploración y permite escapar de óptimos locales más fácilmente

  ¿Cómo se eligen los sucesores cuando no son los mejor valorados (ej., mecanismo de selección de rueda de ruleta como en AG?