

IN7605 Heurísticas para Optimización Entera y Aplicaciones

Clase 3: Metaheurísticas: Aleatorias, Tabu, Genetico

Gonzalo Muñoz, Fernando Ordóñez

1 Septiembre, 2025

Cuando construcción y búsqueda local no encuentran una buena solución y se dispone de más tiempo para resolver el problema, se pueden utilizar una de varias metaheurísticas.

- Métodos aleatorios (Simulated Annealing, VNS, GRASP)
- Aprendizaje en construcción (Hormigas)
- Aprendizaje en búsqueda local (Tabu)
- Métodos poblacionales (Algoritmo Genetico)

Chapter	Method	Operating principles
7. Randomized methods	Simulated annealing	Biased random local search
	Threshold accepting	
	Great deluge	
	Demon algorithm	
	Noising methods	
	Late acceptance hill climbing	Local search + history
8. Construction learning	Variable neighborhood search	Several neighborhoods
	GRASP	Biased random construction + local search
	MIN-MAX ant system	Construction with learning + local search
9. Local search learning	FANT	
	Vocabulary building	Advanced construction learning
	Taboo search	Local search + memory
10. Population management	Strategic oscillations	Local search + various memories
	Genetic algorithm	Simple population evolution
	Memetic algorithm	Advanced population management + local search
	Scatter search	
	BRKGA	
	Path relinking	
	GRASP-PR	
	Fixed set search	

Métodos aleatorios

Algunos aspectos generales, y objetivos

- modificar búsqueda local con componentes aleatorios
- modificar construcción con componentes aleatorios
- permiten considerar soluciones peores en búsqueda de un óptimo global
- tradeoff entre optimizar una búsqueda local (intensificar) y explorar ampliamente la región factible (diversificar)

Simulated Annealing

Temple simulado, recocido simulado, cristalización simulada o enfriamiento simulado

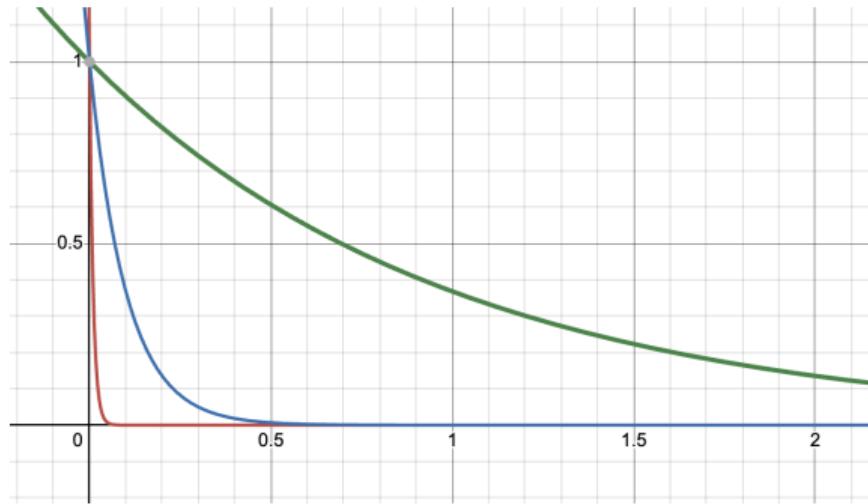
- Inspirado en fenómeno físico en que si el material se enfria lento se puede llegar a una estructura de menor energía (cristal).
- Permite aceptar movimientos m tal que $\Delta = f(s \oplus m) - f(s) > 0$ mientras sea suficientemente pequeño:

$$e^{-\Delta/T} > \text{rand}(0, 1).$$

- Se reduce la temperatura T
 - ▶ para T grande es fácil aceptar un $\Delta > 0$
 - ▶ para $T \sim 0$ el método se transforma en búsqueda local.

Simulated Annealing

$e^{-x/T}$ para $T = 1, 0.1, 0.01$



Simulated Annealing

El siguiente código realiza simulated annealing en la siguiente situación:

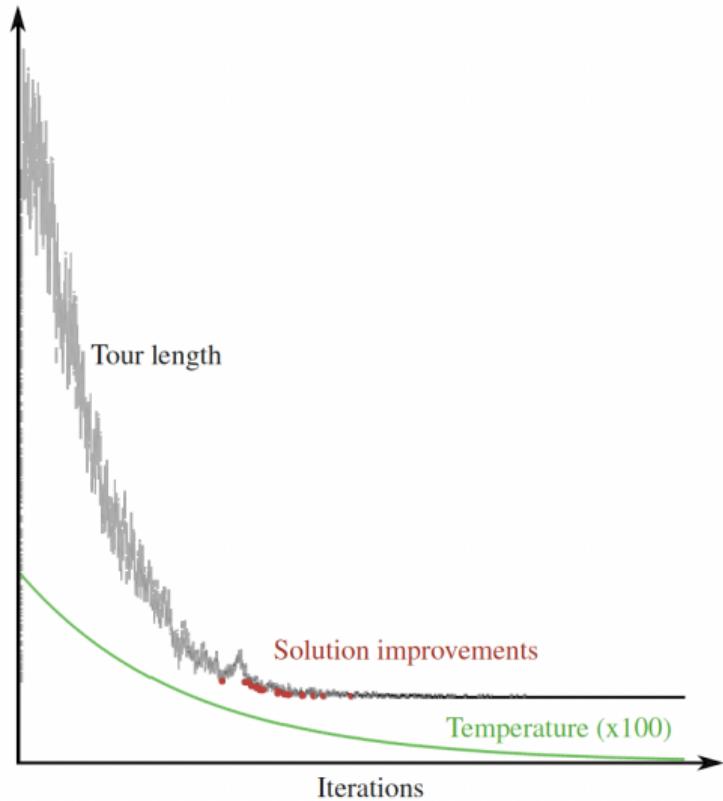
- Dada una solución inicial s
- Modifica una búsqueda local de primera mejora aleatoria
- Dado el parámetro $0 < \alpha < 1$ se actualiza la temperatura T con $T \leftarrow \alpha T$.
- Número de iteraciones por valor de T es 1.

Input: Initial solution s ; fitness function f to minimize; neighborhood structure M ,
parameters $T_{init}, T_{end} < T_{init}$ and $0 < \alpha < 1$

Result: Modified solution s

```
1  $T \leftarrow T_{init}$ 
2 while  $T > t_{end}$  do
3   Randomly generate  $m \in M$ 
4    $\Delta = f(s \oplus m) - f(s)$ 
5   Randomly generate  $0 < u < 1$ 
6   if  $\Delta < 0$  or  $e^{-\Delta/T} > u$  then  $m$  is accepted
7      $s \leftarrow s \oplus m$ 
8    $T \leftarrow \alpha \cdot T$ 
```

Simulated Annealing para el TSP



Simulated Annealing: Variantes

Input: Initial solution s ; fitness function f to minimize; neighborhood structure M ,
parameters $T_{init}, T_{end} < T_{init}$ and $0 < \alpha < 1$

Result: Modified solution s

```
1  $T \leftarrow T_{init}$ 
2 while  $T > t_{end}$  do
3   Randomly generate  $m \in M$ 
4    $\Delta = f(s \oplus m) - f(s)$ 
5   Randomly generate  $0 < u < 1$ 
6   if  $\Delta < 0$  or  $e^{-\Delta/T} > u$  then  $m$  is accepted
7      $s \leftarrow s \oplus m$ 
8    $T \leftarrow \alpha \cdot T$ 
```

- Umbral: $\Delta < \tau_k$
- Gran Diluvio: aceptar mientras $f(s \oplus m) > L$
- Demonio: aceptar mientras $\Delta \leq D$ y actualizar presupuesto con $D \leftarrow D - \Delta$.
- Métodos con ruido: aceptar mientras $\Delta + \text{noise}(k) < 0$

Variable Neighborhood Search

Se consideran distintas vecindades con conjuntos de movimientos M_1, M_2, \dots, M_p para

- intensificación: explorar profundamente usando M_1 en torno a una solución s
- diversificación: cambiar a regiones distintas usando M_2, \dots, M_p aleatoriamente

Input: Solution s , fitness function f to minimize, neighborhood structures $M_1 \dots M_p$

Result: s^*

```
1  $s^* \leftarrow s$ 
2  $k \leftarrow 1$ 
3 while  $k \leq p$  do
4   Randomly generate a move  $m \in M_k$ 
5    $s \leftarrow s \oplus m$ 
6   Find the local optimum  $s'$  associated with  $s$  in neighborhood  $M_1$ 
7   if  $f(s') < f(s^*)$  then
8      $s^* \leftarrow s'$ 
9      $k \leftarrow 1$ 
10  else
11     $s \leftarrow s^*$ 
12     $k \leftarrow k + 1$ 
```

Variable Neighborhood Search para el TSP

Por ejemplo, para el TSP podemos construir las vecindades:

- M_1 : cadenas de Lin-Kernigan
- M_k : permutar k pares de nodos en el tour.

```
1 from random_generators import unif           # Listing 12.1
2 from tsp_utilities import tsp_length          # Listing 12.2
3 from tsp_LK import tsp_LK                     # Listing 12.3
4
5 ##### Variable Neighborhood Search for the TSP
6 def tsp_VNS(d,                                # Distance matrix
7             best_tour,                         # TSP tour
8             best_length):
9
10    n = len(best_tour)
11    iteration, k = 0, 1
12    while k < n:
13        tour = best_tour[:]
14
15        for _ in range(k):                    # Perturbate solution
16            u = unif(0, n - 1)
17            v = unif(0, n - 1)
18            tour[u], tour[v] = tour[v], tour[u]
19            length = tsp_length(d, tour)
20            tour, length = tsp_LK(d, tour, length)
21            iteration += 1
22
23        if length < best_length:
24            best_tour = tour[:]
25            best_length = length
26            print('VNS {:d}\t{:d}\t{:d}'
27                  .format(iteration, k, length))
28
29        k += 1
30
31    return best_tour, best_length
```

GRASP

El *Greedy Randomized Adaptive Search Procedure* (GRASP), repetidamente mejora, usando búsqueda local, una solución obtenida con un método constructivo greedy que incluye componentes aleatorios.

Input: Set E of elements constituting a solution; incremental cost function $c(s, e)$; fitness function f to minimize, parameters I_{max} and $0 \leq \alpha \leq 1$, improvement method *local_search*

Result: Complete solution s^*

```
1  $f^* \leftarrow \infty$ 
2 for  $I_{max}$  iterations do
3   Initialize  $s$  to a trivial partial solution
4    $R \leftarrow E$                                      // Elements that can be added to  $s$ 
5   while  $R \neq \emptyset$  do
6     Find  $c_{min} = \min_{e \in R} c(s, e)$  and  $c_{max} = \max_{e \in R} c(s, e)$ 
7     Choose randomly, uniformly  $e' \in R$  such that
8        $c_{min} \leq c(s, e') \leq c_{min} + \alpha(c_{max} - c_{min})$ 
9        $s \leftarrow s \cup e'$                            // Include  $e'$  in the partial solution  $s$ 
10      Remove from  $R$  the elements that cannot be added any more to  $s$ 
11       $s' \leftarrow \text{local\_search}(s)$            // Find the local optimum associated with  $s$ 
12      if  $f^* > f(s')$  then
13         $f^* \leftarrow f(s')$ 
           $s^* \leftarrow s'$ 
```

- Las iteraciones I_{\max} permiten construir soluciones diferentes
- La construcción es aleatoria
- Encuentra el optimo local (en M_1) de la solución encontrada.

```
1 from random_generators import rand_permutation          # Listing 12.1
2 from tsp_utilities import tsp_length                   # Listing 12.2
3 from tsp_LK import tsp_LK                            # Listing 12.3
4
5 ##### Procedure for producing a TSP tour using GRASP principles
6 def tsp_GRASP(d,                                         # Distance matrix
7               alpha):
8
9     n = len(d[0])
10    tour = rand_permutation(n)
11    for i in range(n - 1):
12        # determine c_min and c_max incremental costs
13        c_min, c_max = float('inf'), float('-inf')
14        for j in range(i + 1, n):
15            if c_min > d[tour[i]][tour[j]]:
16                c_min = d[tour[i]][tour[j]]
17            if c_max < d[tour[i]][tour[j]]:
18                c_max = d[tour[i]][tour[j]]
19
20        next = i+1           # Find the next city to insert, based on lower cost
21        while d[tour[i]][tour[next]] > c_min + alpha * (c_max - c_min):
22            next += 1
23            tour[i + 1], tour[next] = tour[next], tour[i + 1]
24
25        length = tsp_length(d, tour)
26        tour, length = tsp_LK(d, tour, length)
27
28    return tour, length
```

Aprendizaje en Construcción

Los métodos de aprendizaje en construcción tratan de mejorar la mejor solución encontrada sin depender exclusivamente de la incertidumbre. Las técnicas de aprendizaje necesitan los siguientes tres ingredientes.

- Experiencias repetidas y análisis de éxitos y fracasos: sólo se aprende cometiendo errores.
- Memorizando lo que se ha hecho
- Olvidándose de los detalles. Esto nos da la habilidad de generalizar en situaciones similares pero distintas.

Algoritmos de colonia de hormigas

Las hormigas aprenden el camino mas corto a la comida por el aumento de feromonas en el camino.

Comienza como una exploración, los caminos con más feromonas (hormigas vuelven antes) se fortalecen.

El comportamiento de las hormigas que se resume en:

- Una hormiga es un procedimiento (en paralelo) que construye una solución con componentes aleatorios.
- Una huella de feromonas son valores asociados a cada elemento e armando una solución.
- La huella es una memoria colectiva. Despues de construir una solución, los valores de los elementos serán aumentados dependiendo de la calidad de la solución.
- El fenómeno del olvido: evaporación de la huella de feromonas en el tiempo.

Pueden ser abejas, peces, etc.

Algoritmos de colonia de hormigas

El tradeoff entre memoria y función objetivo se logra (para $\alpha > 0$ y $\beta < 0$) seleccionando el siguiente elemento a incluir con una probabilidad proporcional a $\tau_e^\alpha c(s, e)^\beta$.

Algorithm 8.1: MAX-MIN ant system framework

Input: Set E of elements constituting a solution; incremental cost function $c(s, e) > 0$; fitness function f to minimize, parameters $I_{max}, m, \alpha, \beta, \tau_{min}, \tau_{max}, \rho$ and improvement method $a(\cdot)$

Result: Solution s^*

```
1  $f^* \leftarrow \infty$ 
2 for  $\forall e \in E$  do
3    $\tau_e \leftarrow \tau_{max}$ 
4 for  $I_{max}$  iterations do
5   for  $k = 1 \dots m$  do
6     Initialize  $s$  as a trivial, partial solution
7      $R \leftarrow E$  // Elements that can be added to  $s$ 
8     while  $R \neq \emptyset$  do Build a new solution
9       Randomly choose  $e \in R$  with a probability proportional to  $\tau_e^\alpha \cdot c(s, e)^\beta$  // Ant colony formula
10       $s \leftarrow s \cup e$ 
11      From  $R$ , remove the elements that cannot be added any more to  $s$ 
12       $s_k \leftarrow a(s)$  // Find the local optimum  $s_k$  associated with  $s$ 
13      if  $f^* > f(s_k)$  then Update the best solution found
14         $f^* \leftarrow f(s_k)$ 
15         $s^* \leftarrow s_k$ 
16 for  $\forall e \in E$  do Pheromone trail evaporation
17    $\tau_e \leftarrow (1 - \rho) \cdot \tau_e$ 
18    $s_b \leftarrow$  best solution from  $\{s_1, \dots, s_m\}$ 
19   for  $\forall e \in s_b$  do Update trail, maintaining it between the bounds
20      $\tau_e \leftarrow \max(\tau_{min}, \min(\tau_{max}, \tau_e + 1/f(s_b)))$ 
```

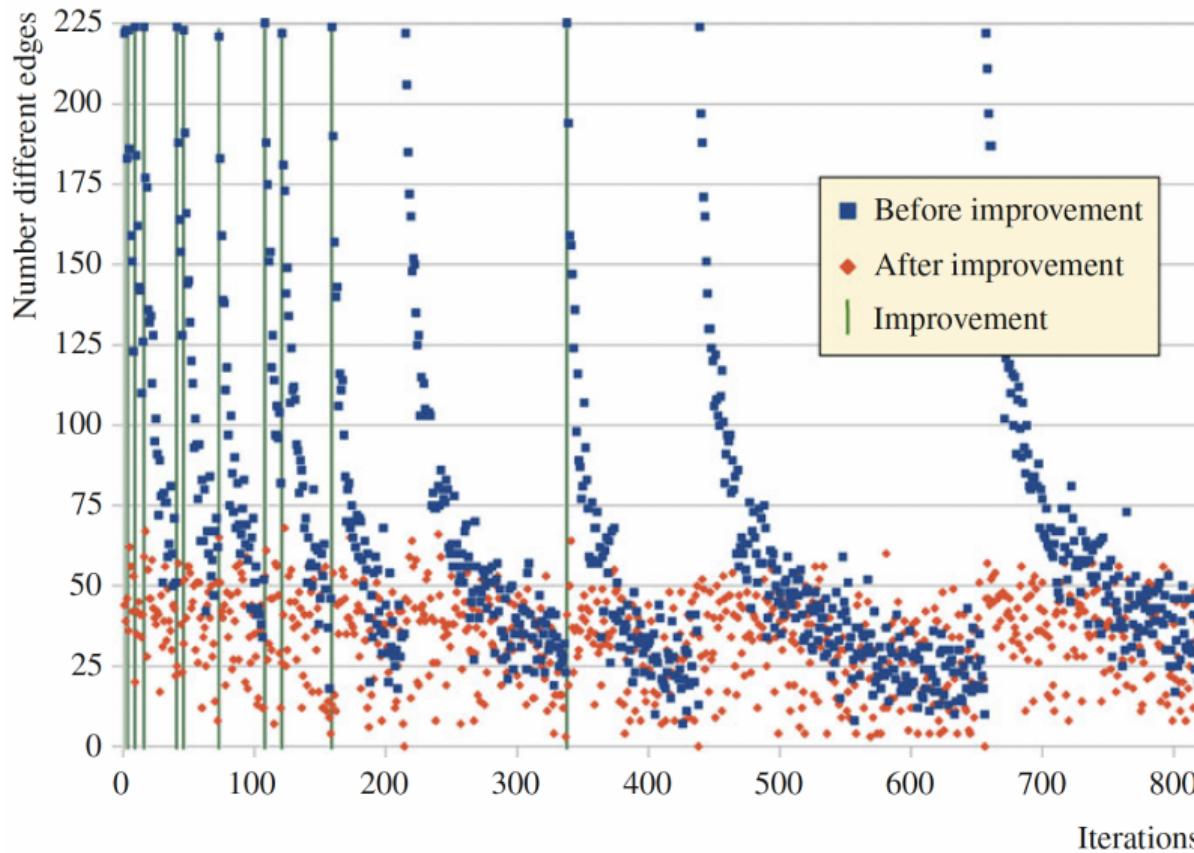
Algoritmos de colonia de hormigas FANT

Input: Set E of elements constituting a solution; fitness function f to minimize,
parameters I_{max}, τ_b and improvement method $a(\cdot)$

Result: Solution s^*

```
1  $f^* \leftarrow$ 
2  $\tau_c \leftarrow 1$ 
3 for  $\forall e \in E$  do
4    $\tau_e \leftarrow \tau_c$ 
5 for  $I_{max}$  iterations do
6   Initialize  $s$  to a partial, trivial solution
7    $R \leftarrow E$                                 // Elements that can be added to  $s$ 
8   while  $R \neq \emptyset$  do
9     Randomly choose  $e \in R$  with a probability proportionnal to  $\tau_e$ 
10     $s \leftarrow s \cup e$ 
11    From  $R$ , remove the elements that cannot be added any more to  $s$ 
12     $s' \leftarrow a(s)$                          // Find the local optimum  $s'$  associated with  $s$ 
13    if  $s' = s^*$  then manage over-learning
14       $\tau_c \leftarrow \tau_c + 1$                   // More weight to the newly constructed solutions
15      for  $\forall e \in E$  do Erase all trails
16         $\tau_e \leftarrow \tau_c$ 
17    if  $f^* > f(s_k)$  then manage best solution improvement
18       $f^* \leftarrow f(s_k)$ 
19       $s^* \leftarrow s_k$                         // Update best solution
20       $\tau_c \leftarrow 1$                       // Give minimum weight to the newly constructed solutions
21      for  $\forall e \in E$  do Erase all trails
22         $\tau_e \leftarrow \tau_c$ 
23    for  $\forall e \in s'$  do reinforce the trails associated with the current solution
24       $\tau_e \leftarrow \tau_e + \tau_c$ 
25    for  $\forall e \in s^*$  do reinforce the trails associated with the best solution
26       $\tau_e \leftarrow \tau_e + \tau_b$ 
```

Algoritmos de colonia de hormigas FANT



Algoritmos de colonia de hormigas FANT

```
1 from random_generators import rand_permutation          # Listing 12.1
2 from generate_solution_trail import *                   # Listing 8.1
3 from init_update_trail import *                         # Listing 8.2
4 from tsp_LK import tsp_LK                             # Listing 12.3
5
6 ##### Fast Ant System for the TSP
7 def tsp_FANT(d,                                         # Distance matrix
8               exploitation,                      # FANT Parameters: global reinforcement
9               iterations):                     # number of solution to generate
10
11    n = len(d[0])
12    best_cost = float('inf')
13    exploration = 1
14    trail = [[-1] * n for _ in range(n)]
15    trail = init_trail(exploration, trail)
16    tour = rand_permutation(n)
17    for i in range(iterations):
18        # build solution
19        tour, cost = generate_solution_trail(d, tour, trail)
20        # improve built solution with a local search
21        tour, cost = tsp_LK(d, tour, cost)
22        if cost < best_cost:
23            best_cost = cost
24            print('FANT {:_d} {:_d}'.format(i+1, cost))
25            best_sol = list(tour)
26            exploration = 1                      # Reset exploration to lowest value
27            trail = init_trail(exploration, trail)
28        else:
29            # pheromone trace reinforcement - increase memory
30            trail, exploration = update_trail(tour, best_sol,
31                                                exploration, exploitation, trail)
32    return best_sol, best_cost
```

Aprendizaje en Busqueda Local

Busquedas locales son esenciales en metaheurísticas. Prácticamente todas las heurísticas eficientes incorporan una búsqueda local. Además, metaheurísticas a veces son definidas como un proceso maestro que guía una búsqueda local.

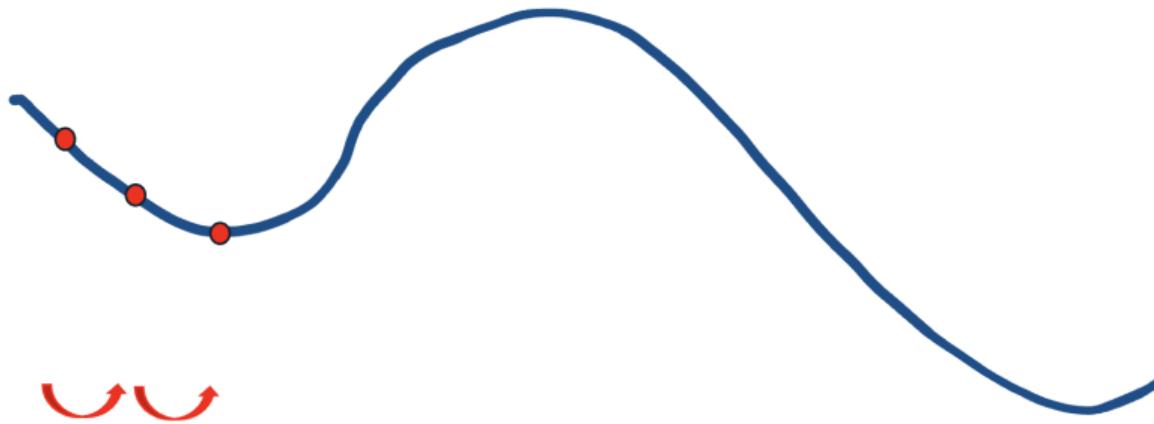
Busqueda Tabu

La idea es

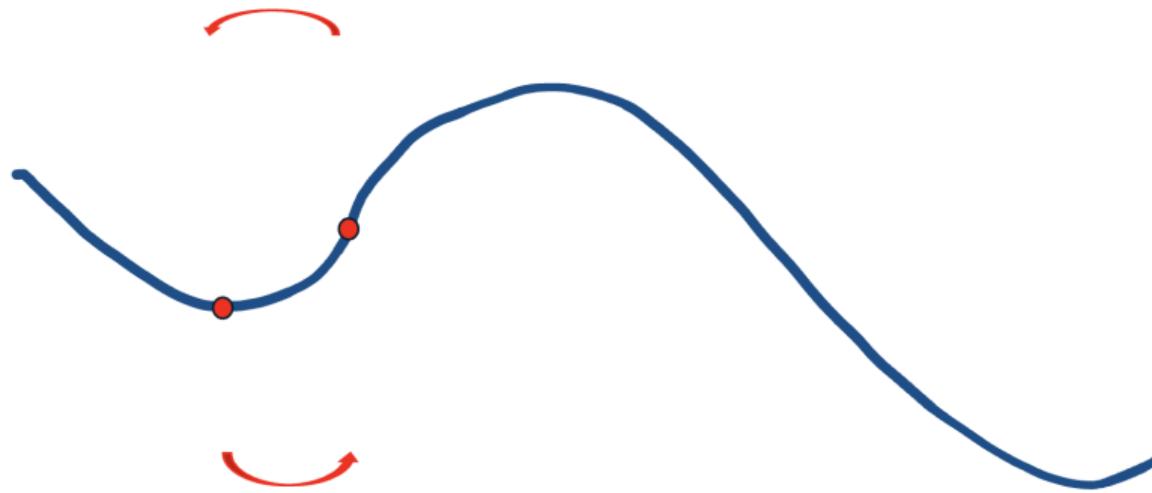
- permitir cambios a soluciones que tengan un peor valor para poder salir de un mínimo local
- evitar, por un tiempo, realizar cambios de deshacer el movimiento realizado.

Se busca evitar repetir una solución ya vista anteriormente.

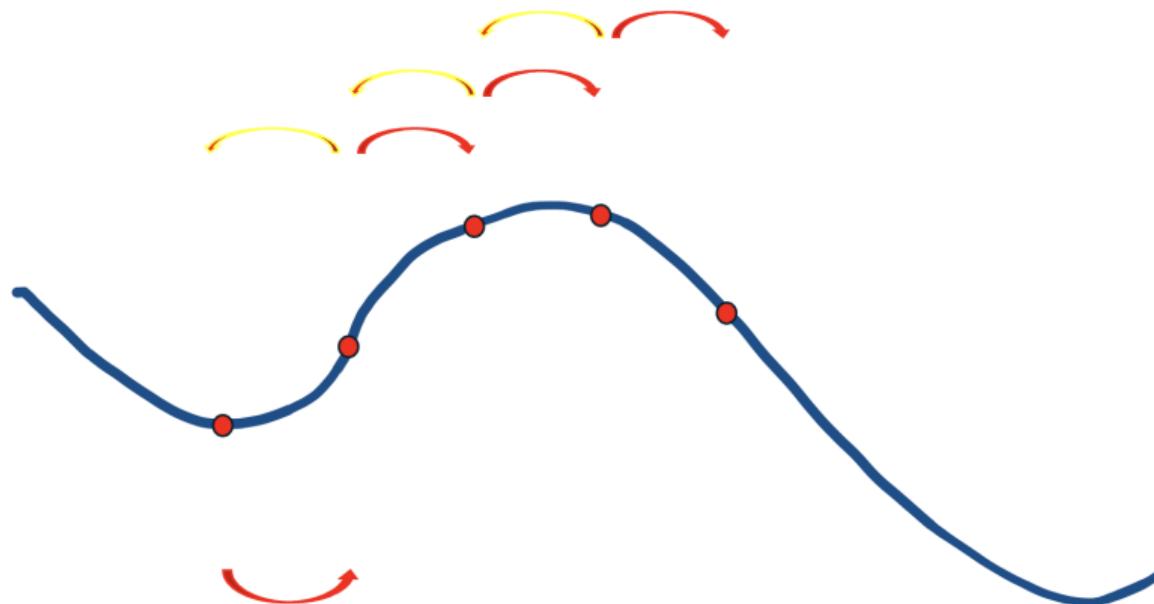
Busqueda Tabú



Busqueda Tabu



Busqueda Tabu



Busqueda Tabu: tabla hash

Una forma eficiente de almacenar las soluciones vistas.

- Un valor entero $h(s)$ se asocia con cada solución s del problema.
- Si visitamos la solución s_i en la iteración i de la búsqueda, guardamos i en el hash $h(s_i) \bmod m$ de un arreglo T con m enteros.
- Entonces, el valor t_k de la k -esima posición de T indica en que iteración una solución con el valor hash k (modulo m) fue visitada.

Busqueda Tabu

Input: Solution s , set M of moves, fitness function $f(\cdot)$ to minimize, parameters I_{max}, d .

Result: Improved solution s^*

```
1  $s^* \leftarrow s$ 
2 for  $I_{max}$  iterations do
3    $best\_neighbor\_value \leftarrow \infty$ 
4   forall  $m \in M$  (such that  $m$  (or  $s \oplus m$ ) is not marked as taboo) do
5     if  $f(s \oplus m) < best\_neighbor\_value$  then
6        $best\_neighbor\_value \leftarrow f(s \oplus m)$ 
7        $m^* \leftarrow m$ 
8   if  $best\_neighbor\_value < \infty$  then
9     Mark  $(m^*)^{-1}$  (or  $s$ ) as taboo for the next  $d$  iterations
10     $s \leftarrow s \oplus m^*$ 
11    if  $f(s) < f(s^*)$  then
12       $s^* \leftarrow s$ 
13  else
14    Error message:  $d$  too large: no move allowed!
```

Busqueda Tabu: movimientos tabu

Ejemplo busqueda tabu

$$\begin{array}{llllllllll} \max & 12s_1 & +10s_2 & +9s_3 & +7s_4 & +4s_5 & +8s_6 & +11s_7 & +6s_8 & +13s_9 \\ \text{s.a.} & 10s_1 & +12s_2 & +8s_3 & +7s_4 & +5s_5 & +13s_6 & +9s_7 & +6s_8 & +14s_9 & \leq 45 \\ & & & & & & & & & & s_i \in \{0, 1\}, i = 1 \dots 9 \end{array}$$

- Soluciones: vectores de 0/1 de dimensión 9, i.e. $s \in \{0, 1\}^9$
- Vecindad: soluciones con una coordenada diferente

$$N(s) = \{s' \mid s'_i = (s_i + (e_j)_i) \bmod 2, e_j \text{ vector canónico}\}$$

- Largo lista tabu: $d = 4$
- Lista: vector de dimensión 9, t_i indica en que iteración puedo modificar ítem i

Busqueda Tabu: movimientos tabu

Iteration number	Variable modified	Modified solution	Fitness value	Volume used	Taboo status
1	s_9	(0, 0, 0, 0, 0, 0, 0, 0, 1)	13	14	(0, 0, 0, 0, 0, 0, 0, 0, 4)
2	s_7	(1, 0, 0, 0, 0, 0, 0, 0, 1)	25	24	(5, 0, 0, 0, 0, 0, 0, 0, 4)
3	s_7	(1, 0, 0, 0, 0, 0, 1, 0, 1)	36	33	(5, 0, 0, 0, 0, 0, 6, 0, 4)
4	s_2	(1, 1, 0, 0, 0, 0, 1, 0, 1)	46	45	(5, 7, 0, 0, 0, 0, 6, 0, 4)
5	s_9	(1, 1, 0, 0, 0, 0, 1, 0, 0)	33	31	(5, 7, 0, 0, 0, 0, 6, 0, 8)
6	s_3	(1, 1, 1, 0, 0, 0, 1, 0, 0)	42	39	(5, 7, 9, 0, 0, 0, 6, 0, 8)
7	s_8	(1, 1, 1, 0, 0, 0, 1, 1, 0)	48	45	(5, 7, 9, 0, 0, 0, 6, 10, 8)
8	s_2	(1, 0, 1, 0, 0, 0, 1, 1, 0)	38	33	(5, 11, 9, 0, 0, 0, 6, 10, 8)
9	s_4	(1, 0, 1, 1, 0, 0, 1, 1, 0)	45	40	(5, 11, 9, 12, 0, 0, 6, 10, 8)
10	s_5	(1, 0, 1, 1, 1, 0, 1, 1, 0)	49	45	(5, 11, 9, 12, 13, 0, 6, 10, 8)

En el TSP: usar tabla $T \in \mathbb{Z}^{n \times n}$ donde T_{ij} indica la iteración a partir de la cual el arco (i,j) puede estar en un tour.

Busqueda Tabu: movimientos tabu

Duración Tabu (largo de lista tabu)

- $d = 0$: Busqueda local. Se queda en mínimos locales
- $d \gg |N(s)|$: Se puede bloquear, todos los movimientos de una vecindad son tabu. No puede hacer nada.
- El tamaño ideal de la lista tabu depende del problema.
- Existen implementaciones con listas tabu dinámicas.
 - ▶ de forma aleatoria entre d_{\min} y $d_{\min} + d\Delta$
 - ▶ si repite una solución entonces $d \leftarrow d + 1$, si no repite una solución en mucho tiempo entonces $d \leftarrow d - 1$

Criterio Aspiración

Si un movimiento en la lista tabu m es tal que $f(s \oplus m) < f(s^*)$ entonces si se acepta.
Pueden existir otros.

Busqueda Tabu: Largo Plazo

La lista tabu permite mantener un registro de corto plazo para poder salir de mínimos locales. Puede ser importante tener una memoria de largo plazo que facilite la búsqueda de otras regiones factibles.

- **Movidas forzadas** Pueden realizarse movidas fuera de la vecindad para mover la solución a una región distinta.
- **Movidas penalizadas** Guardar el número de veces que ciertas movidas son utilizadas para limitar su uso a futuro y así poder salir de cambios locales.
- **Restart** Utilizar otra solución s para comenzar la búsqueda tabu. Funciona bien con métodos de construcción aleatorios.

Algoritmos Evolutivos

Heurísticas hasta aquí consideran una solución que se van mejorando. También FANT, donde feromonas ayudan a construir una solución. Si hormigas cambian, cambia solución.

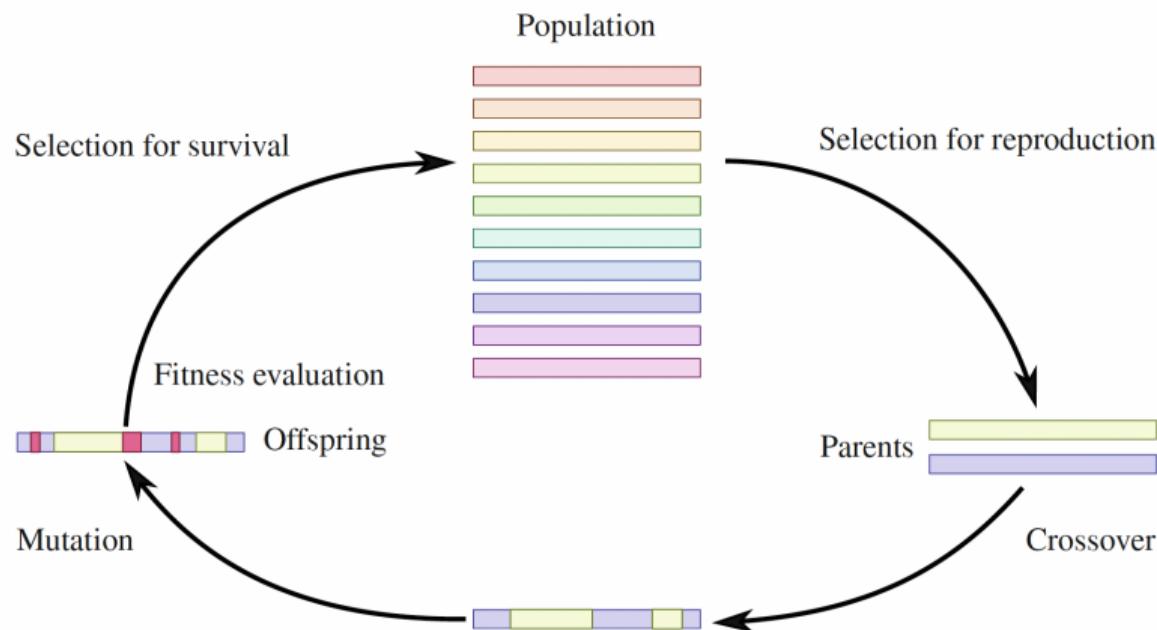
En métodos evolutivos se consideran muchas soluciones (población) que se combina para construir descendientes y nueva población. Son métodos de búsqueda masiva.

Input: Parameters μ and λ , selection for reproduction, crossover, mutation and selection for survival operators

Result: Population of solutions P

- 1 Generate a population P of μ solutions
 - 2 **repeat**
 - 3 Select individuals from P with the selection for reproduction operator
 - 4 Combine the selected individuals with the crossover operator and apply the mutation operator to get λ new solutions
 - 5 Among the $\mu + \lambda$ solutions, select μ individuals with the selection for survival operator; these μ individuals constitute the population P for the next generation
 - 6 **until** a stopping criterion is satisfied
-

Algoritmo Genético



Algoritmo Genético

Selección para reproducción: Se define probabilidad de selección según la calidad de individuo f_i .

- Dado $f_i \geq 0$ la probabilidad de selección es $\frac{f_i}{\sum_{k=1}^{\mu} f_k}$.
- Sampleo con reemplazo: elementos con alta probabilidad seleccionados más de una vez.
- **Selección por rango.** Cada individuo tiene rango r_i , con $r_i = 1$ mejor rango y $r_i = \mu$ el peor. La calidad se calcula $f_i = (1 - \frac{r_i-1}{\mu})^p$. Si $p = 0$ f_i uniforme, $p = 2$ fuertemente guiado por rango.
- **Selección proporcional.** Se construye un f_i a partir de la función objetivo de la solución i en la población.
- **Selección natural.** Seleccionar elementos de la población de forma aleatoria/uniforme. Es tomar $f_i = 1$. La selección de supervivencia debe utilizar criterios de calidad.
- **Selección completa.** Si la población no es muy grande se puede utilizar toda la población en reproducción. La selección de supervivencia debe utilizar criterios de calidad.

Algoritmo Genético

Operador Crossover

Parent 1
9 5 12 8 2 6 1 11 3 10 4 7

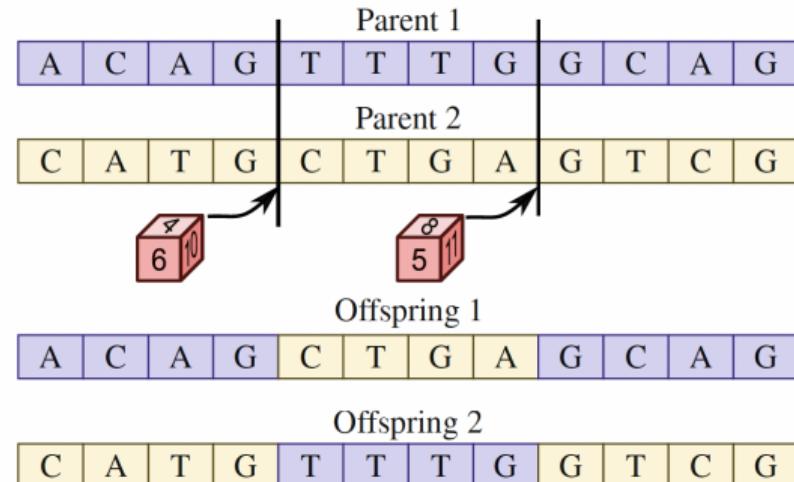
Parent 2
3 11 4 8 12 5 1 2 10 6 9 7



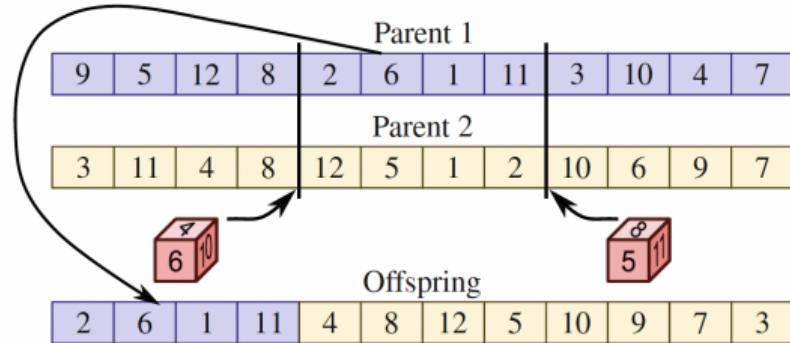
Step 1
3 11 12 8 2 6 1 10 9 7

Offspring
3 11 12 8 2 6 1 5 10 4 9 7

Algoritmo Genético: Crossover



Algoritmo Genético: Crossover



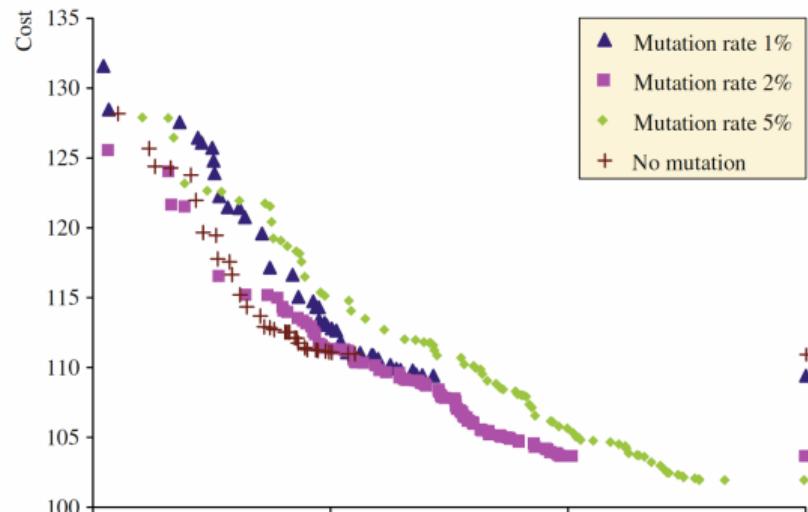
Algoritmo Genético: Mutación

Operador Mutación

Este operador cambia aleatoriamente componentes de un elemento de la población. Esto se puede hacer seleccionando cambios locales de forma aleatoria.

La idea es

- Mejorar la solución mediante cambios locales.
- Aumentar diversidad genética, disminuyendo la velocidad de convergencia del algoritmo.



Algoritmo Genético: Sobrevivencia

Selección para sobrevivencia

- **Reemplazo generacional**

La nueva población es conformada exclusivamente por descendientes.

- **Estrategia evolutiva**

Se generan muchos descendientes $\lambda > \mu$ y se seleccionan los mejores μ descendientes.

- **Reemplazo estacionario.**

Se generan $\lambda = 2$ descendientes que reemplazan a sus padres

- **Reemplazo elitista.**

Se seleccionan los mejores μ elementos de la población y descendientes $\mu + \lambda$

Algoritmo Genético

