Inteligencia Artificial

Búsqueda II

Contenidos

- Ant Colony Optimization.
- Algoritmos Genéticos.
- Búsqueda Tabú.
- Bee Colony Optimization.
- Particle Swarm Optimization

Conceptos Fundamentales

- Una **metaheurística** es un método heurístico para resolver un tipo de problema computacional general, usando los parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera que se espera eficiente. Normalmente, estos procedimientos son heurísticos. El nombre combina el prefijo griego “meta" ("más allá", aquí con el sentido de "nivel superior") y "heurístico" (de ευρισκειν, *heuriskein*, "encontrar").
- Las metaheurísticas generalmente se aplican a problemas que no tienen un algoritmo o heurística específica que dé una solución satisfactoria; o bien cuando no es posible implementar ese método óptimo. La mayoría de las metaheurísticas tienen como objetivo los problemas de optimización combinatoria, pero por supuesto, se pueden aplicar a cualquier problema que se pueda reformular en términos heurísticos.
- Las metaheurísticas no son la panacea y suelen ser menos eficientes que las heurísticas específicas.

Conceptos Fundamentales

- Se habla de heurística para referirse a una técnica, método o procedimiento inteligente de realizar una tarea que no es producto de un riguroso análisis formal, sino de conocimiento experto sobre la tarea.
- En especial, se usa el término heurístico para referirse a un procedimiento que trata de aportar soluciones a un problema con un buen rendimiento, en lo referente a la calidad de las soluciones y a los recursos empleados.
- La idea más genérica del término heurístico está relacionada con la tarea de resolver inteligentemente problemas reales usando conocimiento.

Conceptos Fundamentales

- La concepción más común en IA es interpretar que heurístico es el calificativo apropiado para los procedimientos que, empleando conocimiento acerca de un problema y de las técnicas aplicables, tratan de aportar soluciones (o acercarse a ellas) usando una cantidad de recursos (generalmente tiempo) razonable.
- En un problema de optimización, aparte de las condiciones que deben cumplir las soluciones factibles del problema, se busca la que es óptima según algún criterio de comparación entre ellas.
- En Optimización Matemática (y en investigación de operaciones), el término heurístico se aplica a un procedimiento de resolución de problemas de optimización con una concepción diferente:
  - "*se califica de heurístico a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad o su factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación*"

Conceptos Fundamentales

- En investigación de operaciones, se usa el calificativo heurístico en contraposición a exacto, que se aplica los procedimientos a los que se les exige que la solución aportada sea óptima y factible.
- Una solución heurística de un problema es la proporcionada por un método heurístico, es decir, aquella solución sobre la que se tiene cierta confianzade que es factible y óptima, o de que alcanza un alto grado de optimalidad y/o factibilidad.

Conceptos Fundamentales

- Las metaheurísticas son: estrategias inteligentes para diseñar o mejorar procedimientos heurísticos muy generales con un alto rendimiento.
- Los conceptos actuales de lo que es una metaheurística están basados en las diferentes interpretaciones de lo que es una forma inteligente de resolver un problema.

Conceptos Fundamentales

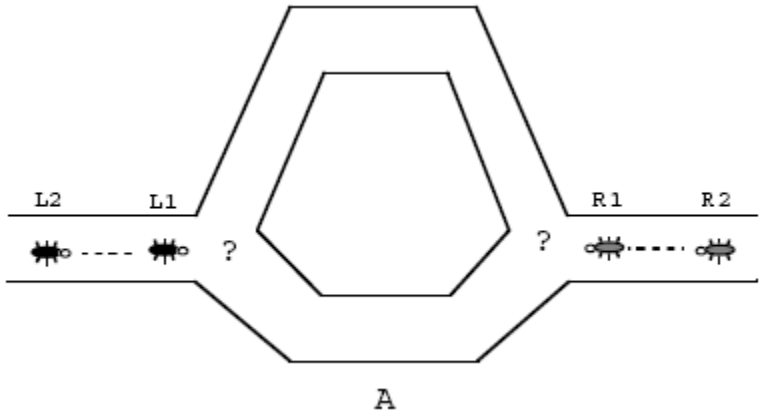
- Hay cuatro tipos de metahuerísticas fundamentales:
  - Las metaheurísticas de **relajación** se refieren a procedimientos de resolución de problemas que utilizan relajaciones d el modelo original (es decir, modificaciones del modelo que hacen al problema más fácil de resolver), cuya solución facilita la solución del problema original.
  - Las metaheurísticas **constructivas** se orientan a los procedimientos que tratan de la obtención de una solución a partir del análisis y selección paulatina de las componentes que la forman.
  - Las metaheurísticas de **búsqueda** guían los procedimientos que usan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entornos asociadas.
  - Las metaheurísticas **evolutivas** están enfocadas a los procedimientos basados en conjuntos de soluciones que evolucionan sobre el espacio de soluciones.

Ant Colony Systems

- Esta meta-heurística se basa en encontrar el camino más corto de una fuente de comida a su nido.
- Mientras andan, las hormigas depositan feromona en el piso, y siguen, con cierta probabilidad, la feromona depositada previamente por otras hormigas.

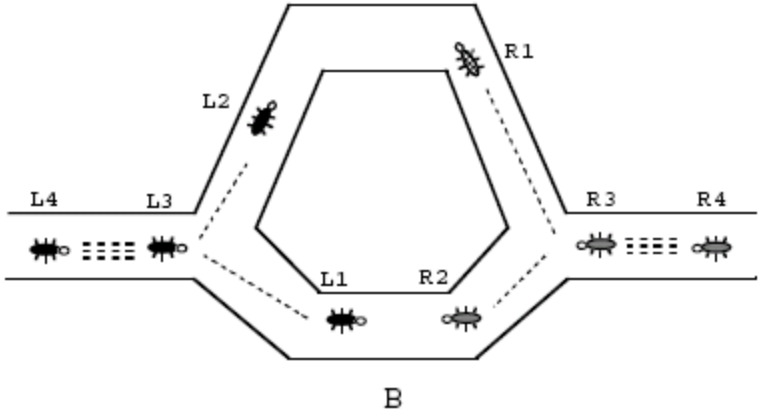
Ant Colony Systems

- Las hormigas llegan a un punto donde tienen que virar a la izquierda o derecha.
- Cuando no tienen idea de qué hacer, es esperable que la mitad de las hormigas van para un lado, y la otra mitad para el otro (aleatoriamente).
- Sucede a ambas, las que se mueven de izquierda a derecha y al revés.



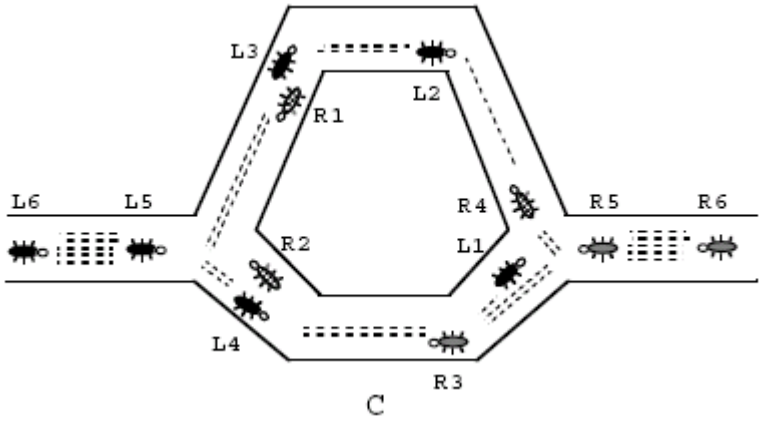
Ant Colony Systems

- La figura muestra lo que sucede después.
- La línea punteada representa la cantidad de feromona derramada por las hormigas.
- Dado que el camino inferior es más corto, más hormigas lo visitaran en promedio.
- La feromona se acumulará más rápido.



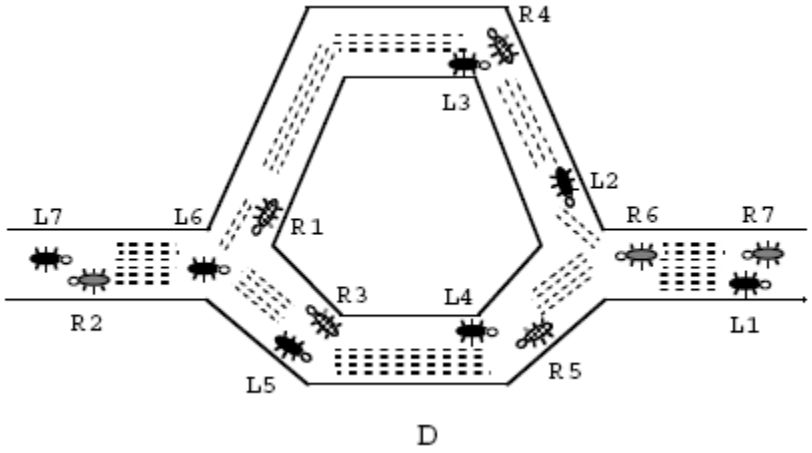
Ant Colony Systems

- Entonces, algunas hormigas escogen el camino superior y otras el inferior.
- Dado que las hormigas se mueven a velocidad casi constante, las hormigas que escogen el camino inferior (más corto) alcanzan el punto de decisión más rápido.
- La feromona se acumula más rápido en el camino más corto.



Ant Colony Systems

- Después de un período de tiempo, la diferencia es lo suficientemente significativa como para influenciar a las nuevas hormigas que entran en el sistema.
- Las hormigas escogen con alguna probabilidad favoreciendo al camino con más feromona.
- Actúa como feedback positivo.



Ant Colony Systems

- ACS está inspirado en este comportamiento de las hormigas.
- La idea es tener agentes (las hormigas) que buscan soluciones buenas para el TSP en paralelo.
- Cooperan entre ellas a través de la feromona, un mecanismo indirecto de comunicación global.

Ant Colony Systems

- Cada hormiga construye una solución de manera iterativa:
  - Agrega ciudades nuevas a una solución parcial mediante la información ganada de la experiencia pasada y una heurística greedy.
- La memoria toma forma de feromona depositada en los arcos.

Ant Colony Systems

- En un inicio *m* hormigas son posicionadas en *n* ciudades de acuerdo a alguna regla de iniciación, e.g., random.
- Cada hormiga construye un tour repitiendo la aplicación de una regla estocástica greedy: la regla de transición de estados.
- Mientras construye su tour, una hormiga modifica también la cantidad de feromona en los arcos visitados mediante la aplicación de una reglas de actualización local.

Ant Colony Systems

- Cuando las hormigas terminan su tour, la cantidad de feromona en los arcos es modificada nuevamente mediante la regla de actualización local.
- Una hormiga *r* escoge una ciudad *s* para moverse de acuerdo a:

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \right\} & \text{if } q \le q_0 \quad \text{(exploitation)} \\ S & \text{otherwise} \quad \text{(biased exploration)} \end{cases}$$

Ant Colony Systems

- Donde q es un número aleatorio uniformemente distribuido entre 0 y 1.
- S es una variable aleatoria seleccionada de acuerdo a la distribución:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases}$$

Ant Colony Systems

- En la ecuación anterior, τ es la feromona, $\eta=1/\delta$, $J_k(r)$ es el set de ciudades que quedan pendiente por visitar para la que está en *r*.
- $\beta>0$ es un parámetro que determina la importancia relativa de la feromona versus la distancia.
- Esta regla favorece nodos conectado por arcos cortos con una gran cantidad de feromona.
- El parámetro q_0 determina la importancia de la explotación vs. exploración.

Ant Colony Systems

- Sólo la hormiga que ha encontrado la mejor solución desde el comienzo de la colonia le es permitido ejecutar la regla de actualización global.
- Es la única hormiga que deposita feromona.
- Esta actualización es al final de cada iteración, es decir, cuando todas las hormigas han terminado sus tours.
- Normalmente se explora la vecindad de la mejor solución.

Ant Colony Systems

- Los niveles de feromona son actualizados mediante la siguiente regla:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau(r,s)$$

$$\Delta\tau(r,s) = \begin{cases} \left(L_{gb}\right)^{-1} & \text{if } (r,s) \in \text{ global - best - tour} \\ 0 & \text{otherwise} \end{cases}$$

Ant Colony Systems

- Donde, $0<\alpha<1$ es un parámetro de decaimiento y L_{gb} es el largo de la mejor solución encontrada desde el comienzo del tour.
- Sólo arcos pertenecientes a la mejor solución reciben un reforzamiento.
- Hay otro tipo de regla de actualización que es opcional: iteration-best, la cual toma el mejor tour de la iteración L_{ib}, y refuerza esos arcos.

Ant Colony Systems

- La diferencia en usar y no usar iteration-best es mínima, por lo que general no se usa.
- La regla de actualización local se ejecuta después que una hormiga ha visitado una ciudad.
- Lo que es lo mismo que hacerlo al final que se ha construido la solución. Esta actualización depende de un parámetro $0<\rho<1$:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\Delta\tau(r,s)$$

Ant Colony Systems

- El parámetro $\Delta\tau(r,s)$ puede ser instanciado de varias formas:
  - La fórmula de Q-learning: γ max $\tau(s,z)$, $z \in J_k(s)$
  - El nivel inicial de feromona: τ_0
  - Utilizar 0.
  - Sin regla de actualización.
- La dos primeras reglas han mostrado ser las mejores experimentalmente, siendo la segunda la menos costosa computacionalmente, por eso es la que se usa.

Ant Colony Systems

- El conjunto de parámetros convencionales que utiliza ACS está dado por:
  - $\beta=2$
  - $q_0=0.9$
  - $\alpha=\rho=0.1$
  - $\tau_0= 1/(nL_{nn})$
  - $m=10$
- L_{nn} es el largo del tour producido por la heurística del vecino más cercano.

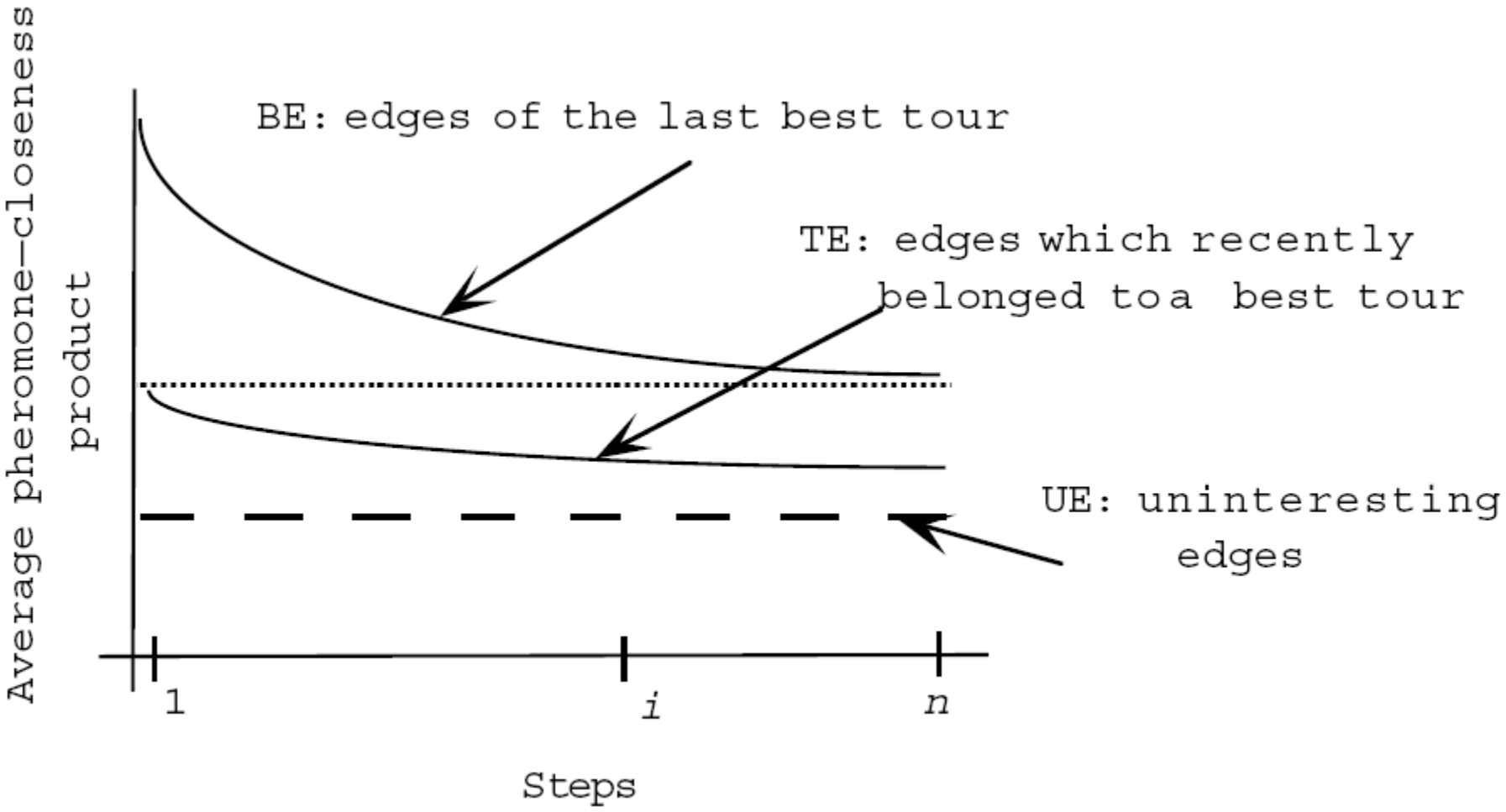
Ant Colony Systems

TABLE I

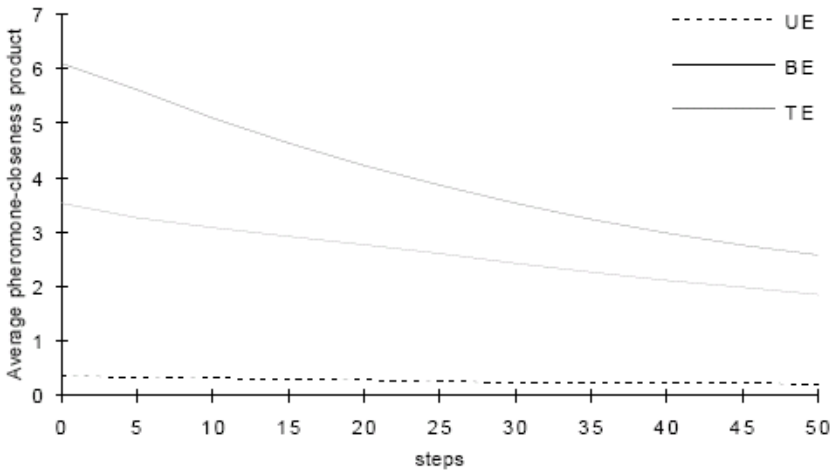
A comparison of local updating rules. 50-city problems and Oliver30 were stopped after 2,500 iterations, while ry48p was halted after 10,000 iterations. Averages are over 25 trials. Results in bold are the best in the Table.

	ACS			Ant-Q			ACS with $\Delta\tau(r,s)=0$			ACS without local-updating		
	average	std dev	best	average	std dev	best	average	std dev	best	average	std dev	best
City Set 1	**5.88**	0.05	**5.84**	**5.88**	0.05	**5.84**	5.97	0.09	5.85	5.96	0.09	**5.84**
City Set 2	**6.05**	0.03	**5.99**	6.07	0.07	**5.99**	6.13	0.08	6.05	6.15	0.09	6.05
City Set 3	**5.58**	0.01	**5.57**	5.59	0.05	**5.57**	5.72	0.12	**5.57**	5.68	0.14	**5.57**
City Set 4	**5.74**	0.03	**5.70**	5.75	0.04	**5.70**	5.83	0.12	**5.70**	5.79	0.05	5.71
City Set 5	**6.18**	0.01	**6.17**	**6.18**	0.01	**6.17**	6.29	0.11	**6.17**	6.27	0.09	**6.17**
Oliver30	424.74	2.83	**423.74**	**424.70**	2.00	**423.74**	427.52	5.21	**423.74**	427.31	3.63	423.91
ry48p	**14,625**	142	**14,422**	14,766	240	**14,422**	15,196	233	14,734	15,308	241	14,796

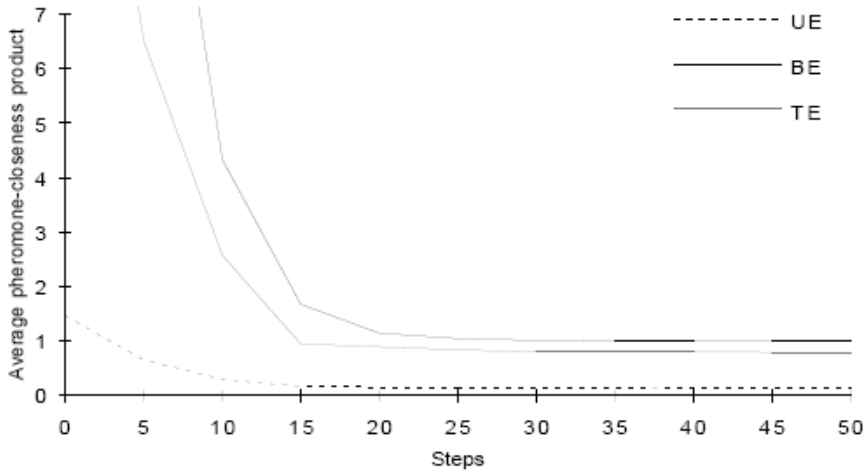
Ant Colony Systems



Ant Colony Systems



a)



b)

Fig. 5. Families of edges classified according to different behavior with respect to the level of the pheromone–closeness product. Problem: Eil51 [14]. a) Pheromone–closeness behavior when the system performance is good. Best solution found after 1,000 iterations: 426, α=ρ=0.1. b) Pheromone–closeness behavior when the system performance is bad. Best solution found after 1,000 iterations: 465, α=ρ=0.9.

Ant Colony Systems

- El número de hormigas que ha mostrado funcionar bien es m = 10.
- ¿Qué pasa si se hacen las hormigas “ciegas” a la feromona?
  - Se puede analizar el comportamiento o la influencia de la colaboración entre hormigas.

Ant Colony Systems

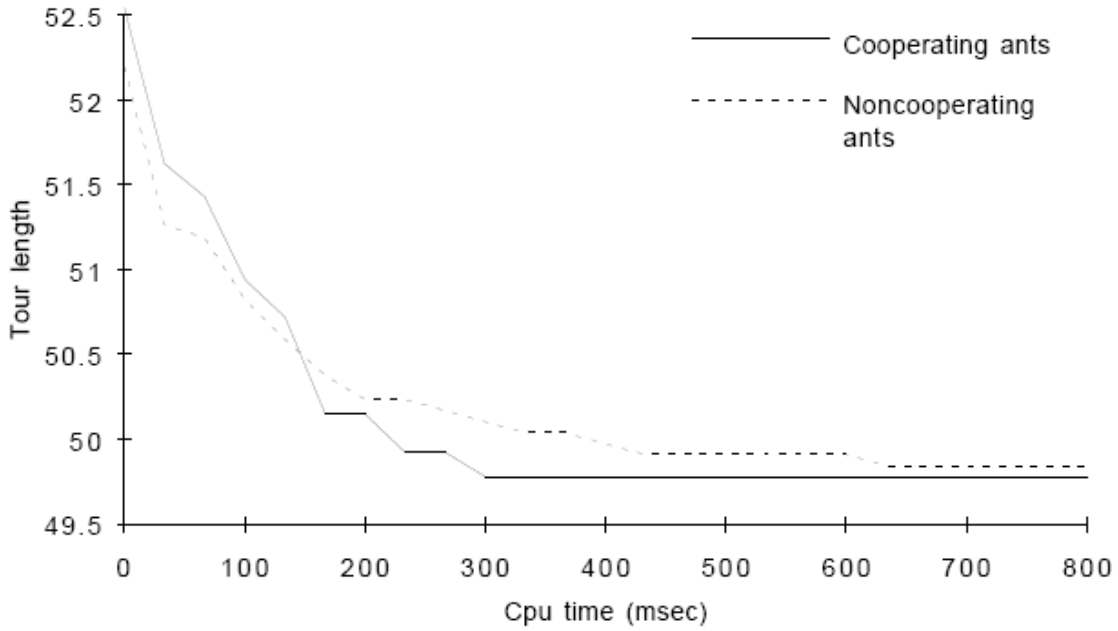


Fig. 8. Cooperating ants find better solutions in a shorter time. Test problem: CCAO [21]. Average on 25 runs. The number of ants was set to m=4.

Ant Colony Systems

TABLE III

Comparison of ACS with other heuristics on geometric instances of the symmetric TSP. We report the best integer tour length, the best real tour length (in parentheses) and the number of tours required to find the best integer tour length (in square brackets). N/A means "not available." In the last column the optimal length is available only for integer tour lengths.

Problem name	ACS	GA	EP	SA	Optimum
Eil50	**425**	428	426	443	**425**
(50-city problem)	(427.96)	(N/A)	**(427.86)**	(N/A)	(N/A)
	[1,830]	[25,000]	[100,000]	[68,512]	
Eil75	**535**	545	542	580	**535**
(75-city problem)	**(542.37)**	(N/A)	(549.18)	(N/A)	(N/A)
	[3,480]	[80,000]	[325,000]	[173,250]	
KroA100	**21,282**	21,761	N/A	N/A	**21,282**
(100-city problem)	**(21,285.44)**	(N/A)	(N/A)	(N/A)	(N/A)
	[4,820]	[103,000]	[N/A]	[N/A]	

Ant Colony Systems

- En el caso de problemas grande, es necesario utilizar una lista de ciudades candidatas ordenados por distancias. Normalmente, se utilizan 15 ciudades.
- Cuando todas las ciudades de esa lista están visitadas, se utiliza la lista normal.

Ant Colony Systems

1. /* Fase inicial*/
   - For each pair (r,s)
     - $\tau(r,s) := \tau_0$
   - End-for
   - For k:=1 to m do
     - Let r_{k1} be the starting city for ant k
     - $J_k(r_{k1}) := \{1, ..., n\} - r_{k1}$
     - /* $J_k(r_{k1})$ es el conjunto de ciudades a ser visitadas por la hormiga k en la ciudad r_{k1}*/
     - $r_k := r_{k1}$ /* r_k es la ciudad donde la hormiga k está ubicada*/
   - End-For

Ant Colony Systems

2. /* Esta es la fase en la cual las hormigas construyen sus tours. El tour de la hormiga se almacena en el $Tour_k$. */
   - For i:=1 to n do
     - If i<n
       - For k:=1 to m do
         - » Escoger la próxima ciudad s_k
         - » $J_k(s_k) := J_k(r_k) - s_k$
         - » $Tour_k(i) := (r_k, s_k)$
       - End-for
     - Else
       - For k:=1 to m do
       - /* Las hormigas regresan a r_{k1} */
         - » $s_k := r_{k1}$
         - » $Tour_k(i) := (r_k, s_k)$
       - End-for
     - End-if
     - /* Actualización de la feromona local*/
     - For k:=1 to m do
       - $\tau(r_k, s_k) := (1-\rho)\tau(r_k, s_k) + \rho\tau_0$
       - $r_k := s_k$ /* nueva ciudad para la hormiga k */
     - End-for
   - End-for

Ant Colony Systems

3. /* Acá se actualiza la feromona de forma global*/
   - For k:=1 to m do
     - Compute L_k /* L_k is el largo del tour hecho por la hormiga k*/
   - End-for
   - Compute L_{best}
   - /*Actualizar los arcos que pertenecen a la mejor solución L_{best} */
   - For each edge (r,s)
     - $\tau(r_k, s_k):=(1-\alpha)\tau(r_k, s_k)+ \alpha (L_{best})^{-1}$
   - End-for
4. If (End_condition = True) then Print shortest o f L_k else goto Phase 2

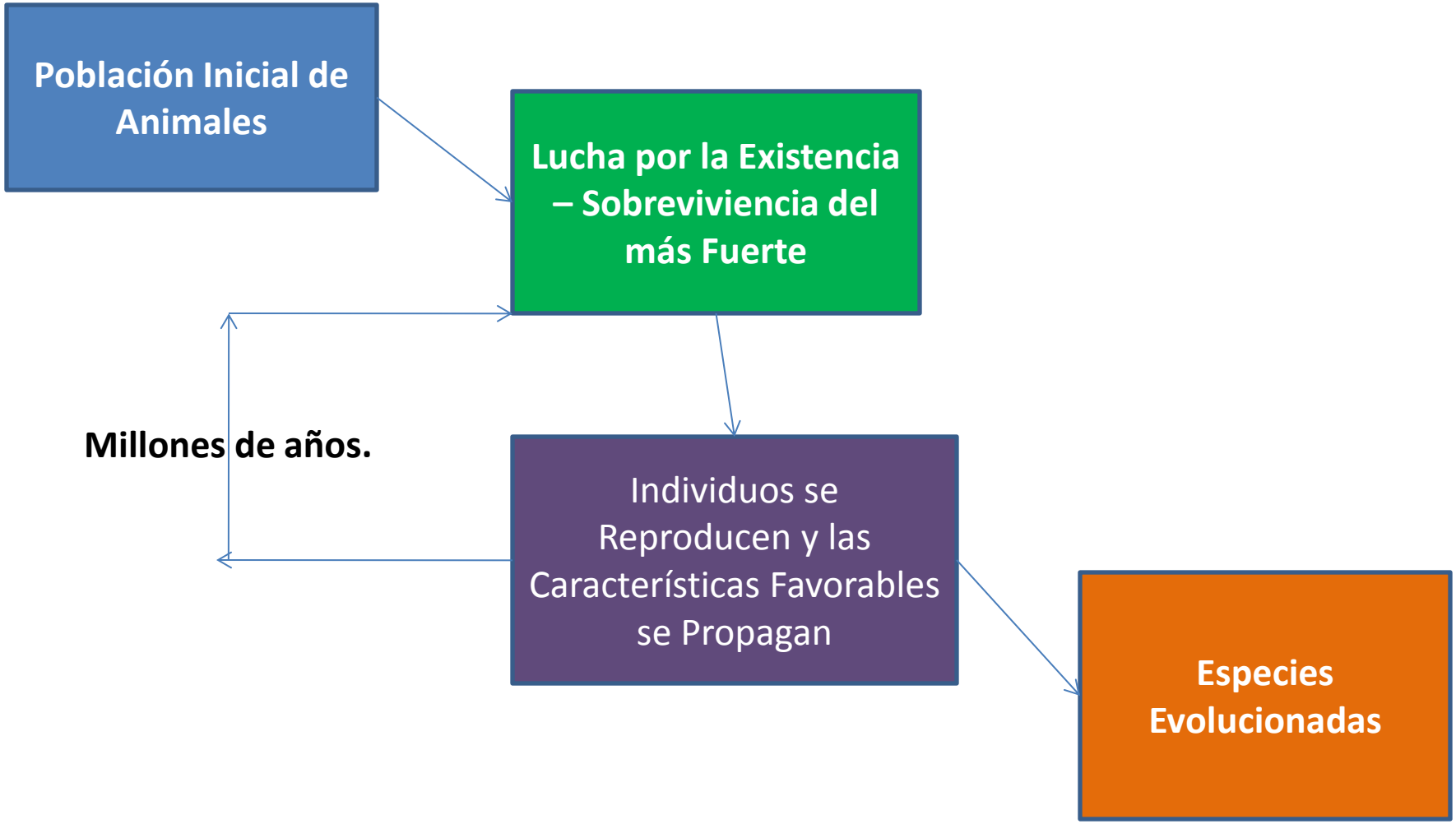
Algoritmos Genéticos

- Son algoritmos de búsqueda (clasificación también) basados en el principio de selección natural.
- El principio de Darwin sobre la selección natural:
  - Si hay organismos que se reproducen, y
  - Si los hijos heredan rasgos de sus progenitores, y
  - Si hay variabilidad en esos rasgos, y
  - Si el ambiente no ayuda a todos los miembros de una población creciente.
  - Entonces, miembros de la población con rasgos menos atractivos para el ambiente morirán.
  - Entonces, miembros con rasgos más atractivos sobrevivirán.

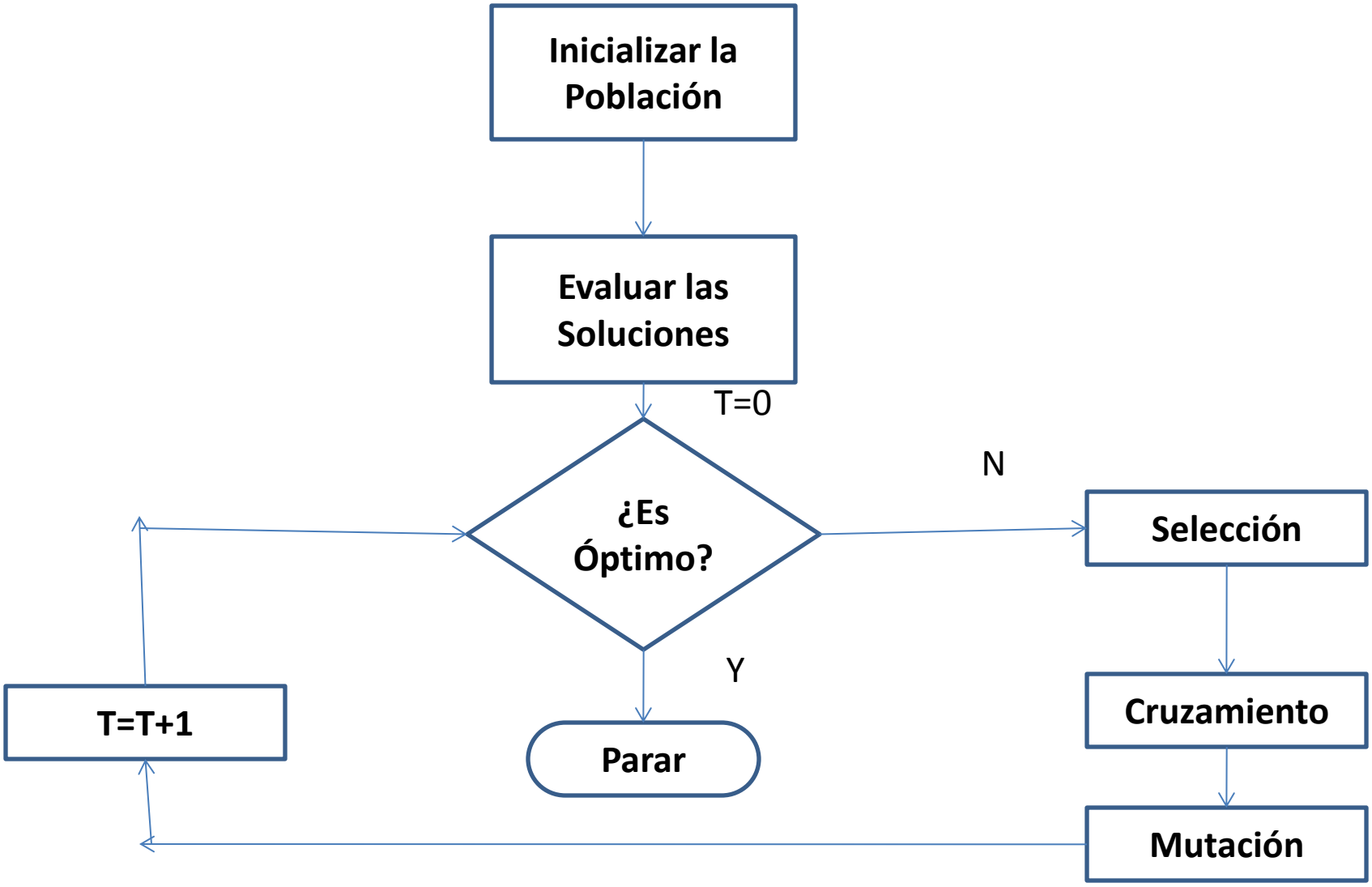
Algoritmos Genéticos

- El resultado de este proceso es la evolución de las especies.
- La idea es "*selecciona lo mejor, elimina el resto*".
- Por ejemplo: las jirafas, el oso hormiguero.

Algoritmos Genéticos



Algoritmos Genéticos



Algoritmos Genéticos

Naturaleza	Computación
Población	Conjunto de soluciones
Individuo	Solución al problema
Fitness	Calidad de la solución
Cromosoma	Codificación de la solución
Gen	Parte de la codificación
Reproducción	Cruzamiento

Algoritmos Genéticos

- La codificación consiste en representar la solución en forma de string de tal manera que contenga la información necesaria para resolver el problema.
- Hay varios tipos: binario, permutación, de valor.

Cromosoma A	**01010101011111010101010101010101**
Cromosoma B	11111010101010100101010101011101010

Algoritmos Genéticos

- Permutación es útil para problemas de ordenamiento como el TSP.

Cromosoma A	0123456789
Cromosoma B	0246813579

Algoritmos Genéticos

- Codificación de valor: Se usa para problemas donde hay valores reales y la representación binaria no es suficiente.
- Normalmente se necesitan operadores especializados.

Cromosoma A	1.235	5.323	0.454	2.321	2.454
Cromosoma B	(left)	(back)	(left)	(right)	(forward)

Algoritmos Genéticos

- La función objetivo cuantifica la optimalidad de una solución tal que una solución pueda ser comparada con otras.
- La idea es que sean rápidas de calcular.
- Se correlaciona con el objetivo.
- En el TSP es la suma de los costos de los caminos entre las ciudades visitadas.

Algoritmos Genéticos

- Recombinación es el proceso que determina que soluciones son preservadas y se permiten reproducir y cuales deben morir.
- La idea es dar énfasis a las soluciones buenas, eliminar las malas, manteniendo el tamaño de la población constante.
- Eliminando las malas soluciones permite tener múltiples copias de las buenas en la población.

Algoritmos Genéticos

- La ruleta es un mecanismo para decidir qué individuos pasan a la siguiente generación.
- Se gira la ruleta n veces, con n el tamaño de la población.
- El individuo escogido pasa a la siguiente generación.
- Nótese: que las soluciones que son mejores tendrán un slot mayor en la ruleta, por ende, mayores oportunidades de ser escogidos.

Algoritmos Genéticos

No.	String	Fitness	%
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		**1170**	**100.0**

Algoritmos Genéticos

- Cruzamiento es el proceso en el cual dos cromosomas combinan su material genético (e.g., bits), para producir su descendencia.
- La descendencia posee características de los padres.
- Dos, o en realidad pueden ser más, individuos son escogidos aleatoriamente del pool de soluciones.
- La forma de cruzamiento depende de la codificación.

Algoritmos Genéticos

- El cruzamiento de punto simple consiste en escoger un punto aleatorio e intercambiar el material genético:

Cromosoma A	**0101010101111101 0101010101010101010**
Cromosoma B	1111101010101010 0101010101011101010

Cromosoma A'	**0101010101111101010101010101011101010**
Cromosoma B'	1111101010101010010101010101010101010

Algoritmos Genéticos

- El cruzamiento de dos puntos consiste en escoger dos puntos aleatorios e intercambiar el material genético:

Cromosoma A	**0101010101111010101010101010101010**
Cromosoma B	1111101010101010010101010101011101010

Cromosoma A'	**1111101010111010101010101011101010**
Cromosoma B'	0101010101101010010101010101010101010

Algoritmos Genéticos

- En general, el cruzamiento no necesariamente produce mejores hijos, o igualmente buenos.
- Si los padres son buenos, la probabilidad de que los hijos sea alta también es buena.
- Si los hijos son malos, lo más probable es que sean eliminados en las próximas generaciones.
- **Epítasis:** es bueno diseñar bloques de variables correlacionadas.

Cruzamiento vs. Mutación

- Se dice que el cruzamiento es un operador m-ario porque utiliza m soluciones para generar las nuevas soluciones.
  - Nótese que el número de soluciones generadas no necesariamente tienen que ser “m”.
  - Es utilizado con algoritmos que manejan poblaciones de soluciones.
- El operador de mutación es un operador unario, dado que de una solución genera normalmente otra única solución.

Algoritmos Genéticos

- **Elitismo** es el método que copia el mejor cromosoma a la nueva población antes del cruzamiento y la mutación.
- Durante estos dos operadores la mejor solución puede perderse.
- En la práctica, elitismo mejora los resultados significativamente.

Algoritmos Genéticos

- La idea de la mutación es cambiar un string deliberadamente de manera de mantener la diversidad en la población.
- La probabilidad de mutación es la que establece cuán frecuentemente se cambian partes de un cromosoma.

Cromosoma A	**01010101011111010101010101010101010**
Offspring	011101010111110**1**101010101010**1**1010

Algoritmos Genéticos

- El hecho de que los algoritmos genéticos lleven a cabo una búsqueda mediante varios puntos es importante.
  - Es una búsqueda paralela.
- Pueden examinar varios peaks simultáneamente reduciendo la probabilidad de quedar estancado en un mínimo local.
- Es un método ciego, y puede ser usado con … cualquier función objetivo.

Algoritmos Genéticos

- Es fácil de paralelizar.
- Hacen uso de una transición probabilística a través de las iteraciones.
- Son robustos y pueden ser aplicados a una gran gama de problemas.

Algoritmos Genéticos

- Supongamos el siguiente tour de seis ciudades: {1,2,3,4,5,6}.
- Es un ejemplo de codificación de permutaciones ya que la función objetivo depende del orden de los elementos.

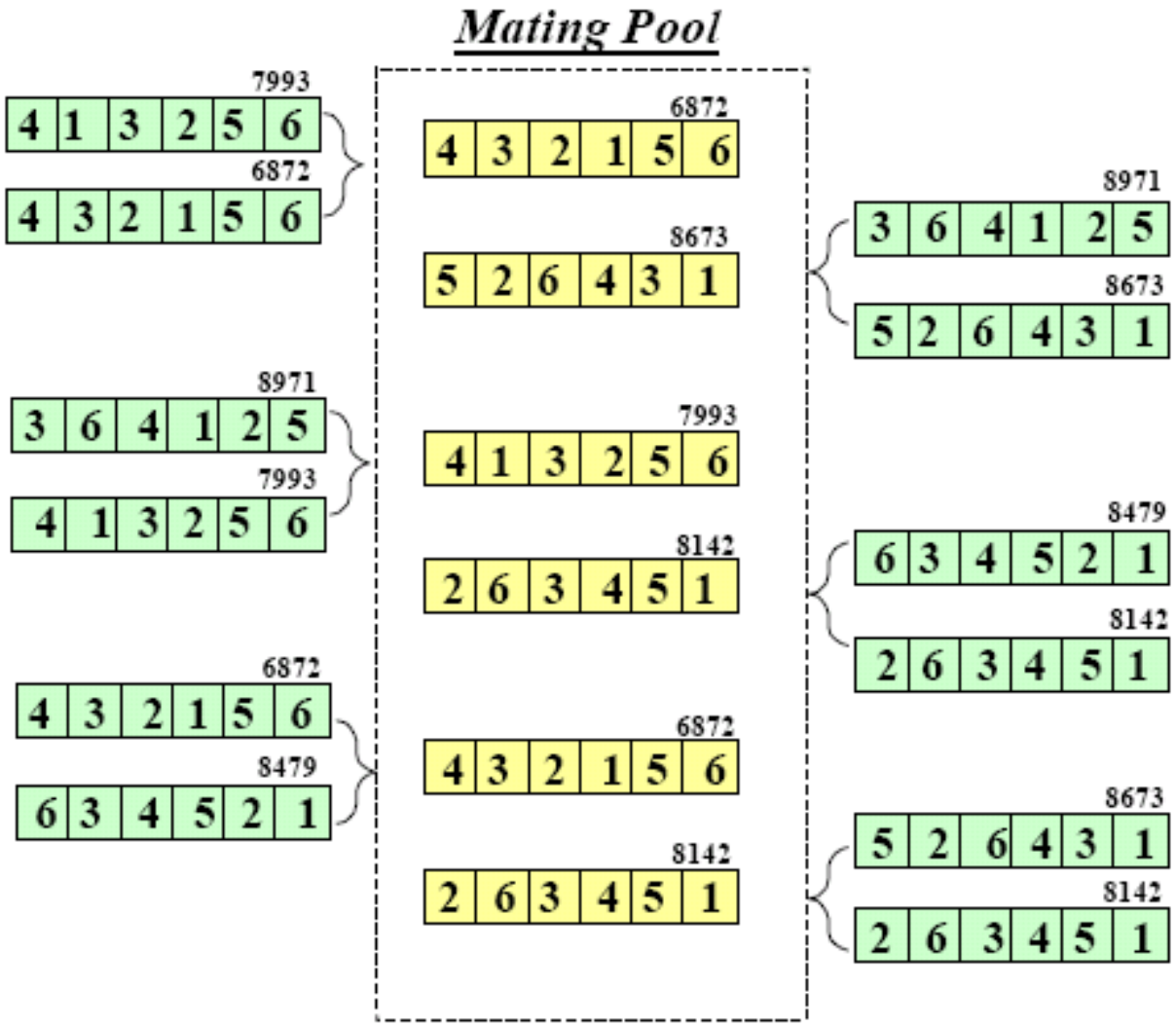
Algoritmos Genéticos

	1	2	3	4	5	6
1	0	863	1987	1407	998	1369
2	863	0	1124	1012	1049	1083
3	1987	1124	0	1461	1881	1676
4	1407	1012	1461	0	2061	2095
5	998	1049	1881	2061	0	331
6	1369	1083	1676	2095	331	0

Algoritmos Genéticos

- La función objetivo para el tour {4,1,3,2,5,6} sería:
  - Fitness = 1407 + 1987 + 1124 + 1049 + 331 + 2095 = 7993 kms.
- Para escoger los individuos para cruzar y mutar se puede utilizar la selección por torneo.
  - Se escoge dos soluciones al azar y torneos se juegan entre ellas para ver quien pasa al “mating pool”.

Algoritmos Genéticos



Algoritmos Genéticos

- Uno de los problemas con la codificación de permutación es el cruzamiento.
- Por ejemplo, si los padres {4,1,3,2,5,6} y {4,3,2,1,5,6} son cortados por la mitad, tenemos los hijos {4,1,3,1,5,6} y {4,3,2,2,5,6}.
- Es decir, produce soluciones inválidas.
- Hay operadores especializados: "*Enhanced Edge Recombination*"

Algoritmos Genéticos

- La idea es generar una tabla de adyacencias en las que se modelan los arcos que salen y entran a una ciudad.
- Arcos comunes poseen un signo negativo.
- Por ejemplo, los padres {4,1,3,2,5,6} y {4,3,2,1,5,6}:

Tabla de adyacencia				
1	4	3	2	5
2	-3	5	1	
3	1	-2	4	
4	-6	1	3	
5	1	2	-6	
6	-5	-4		

Algoritmos Genéticos

1. Escoger alguna ciudad inicial de alguno de los dos padres. Puede ser aleatoriamente. Esta es la “ciudad actual”.
2. Remover todas las instancias de la “ciudad actual” del lado izquierdo de la tabla.
3. Si habían arcos, ir al paso 4 sino al 5.
4. Determinar qué ciudad en la lista de la “ciudad actual” tiene la menor cantidad de entradas en su propia lista. La que tiene menos se convierte en la ciudad actual. En caso de que haya un negativo, se le da preferencia. Empates se quiebran aleatoriamente. Ir al paso 2.
5. Si no hay ciudades no-visitadas, para, de lo contrario escoger una aleatoria e ir al paso 2.

Algoritmos Genéticos

Tabla de adyacencia				
1	4	3	2	5
2	-3	5	1	
3	1	-2	4	
4	-6	1	3	
5	1	2	-6	
6	-5	-4		

Tabla de adyacencia				
1	-	3	2	5
2	-3	5	1	
3	1	-2	-	
4	-6	1	3	
5	1	2	-6	
6	-5	--		

Tabla de adyacencia				
1	-	3	2	5
2	-3	5	1	
3	1	-2	-	
4	-	1	3	
5	1	2	-	
6	-5			

Algoritmos Genéticos

Tabla de adyacencia				
1	-	3	2	-
2	-3	-	1	
3	1	-2	-	
4	-	1	3	
5	1	2	-	
6	-			

Tabla de adyacencia				
1	-	3	2	-
2	-3	-	-	
3	-	-2	-	
4	-	-	3	
5	-	2	-	
6	-			

Tabla de adyacencia				
1	-	-	2	-
2	-	-	-	
3	-	-2	-	
4	-	-	-	
5	-	2	-	
6	-			

Tour resultante: {4,6,5,3,2,1}

Algoritmos Genéticos

- El operador de mutación introduce un cambio en la solución de manera de mantener la diversidad en la población.
- Así se previene la convergencia prematura.
- En el caso del TSP es muy fácil, ya que basta intercambiar dos ciudades.
- Por ejemplo, si se muta {4,1,3,2,5,6} a {4,5,3,2,1,6}

Tabu Search

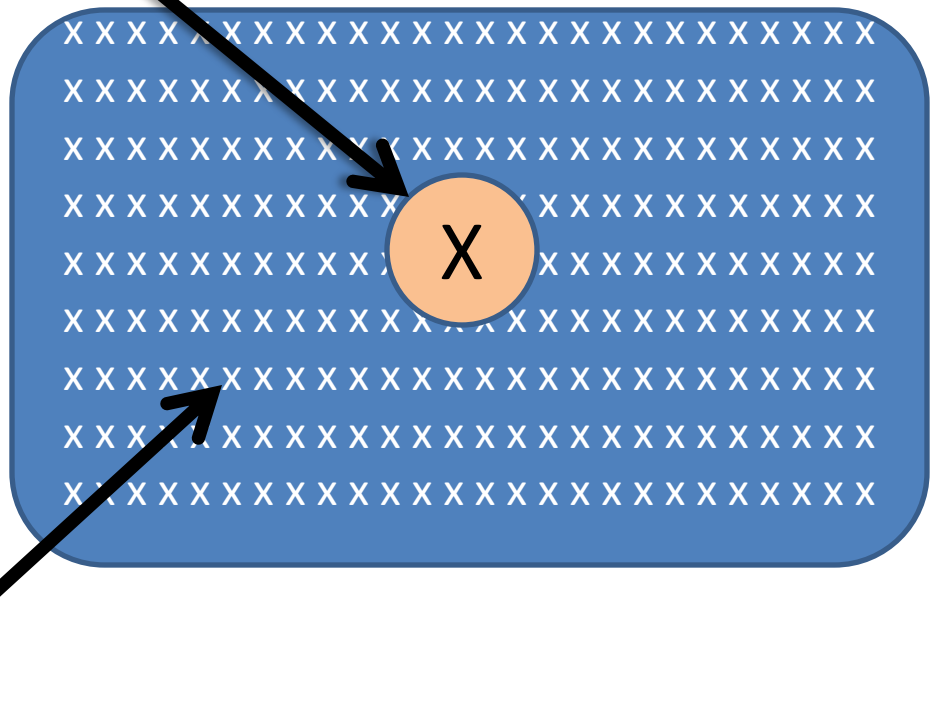
- *“Es mejor una mala decisión basada en información que una buena decisión al azar, ya que, en un sistema que emplea memoria, una mala decisión basada en una estrategia proporcionará claves útiles para continuar la búsqueda. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones*.”

Tabu Search

- Memoria Adaptativa: selecciona que recordar y como olvidar.
- Exploración sensible: intenta buscar buenas características en las soluciones.
- Recordar que los algoritmos genéticos no tienen memoria …. O si?

Tabu Search

Solución Actual X



Vecindario N(X) de X

Tabu Search

- El proceso parte con una solución X escogida al azar.
- El proceso va pasando iterativamente a la mejor solución del vecindario N(X). ¿Se parece bastante a un hill-climbing, cierto?
- Siempre se acepta un movimiento aunque sea peor. El problema es que puede volver repetidamente a la solución anterior!
- Condiciones de término, un número determinado de iteraciones o algún criterio de calidad de solución.

Tabu Search

- La idea es utilizar $N^*(X) \subset N(X)$, donde la lista tabú es $N(X) - N^*(X)$.
- La lista tabú puede ser una memoria de corto plazo, es decir el conjunto de soluciones visitadas en un pasado cercano.
- La memoria puede ser explícita y guardar soluciones completas {1,2,3,4,5,6}, o bien, atributiva, es decir guardar patrones {1,2,3,*,*,*}

Tabu Search

- La lista tabú puede ser dinámica porque después de un número de iteraciones la búsqueda estará en otro lado del espacio.
- El uso de la memoria atributiva puede ser malo ya que puede prohibir moverse a algún punto interesante que no ha sido visitado.
- Se puede relajar la prohibición: sólo se rechaza si su valor no alcanza un nivel de aspiración A.
  - Umbral escogido por el decidor
  - La mejor solución hasta el momento.

Tabu Search

- Hay diferentes tipos de memoria: de corto y largo plazo.
  - **Corto Plazo:** Utiliza (atributos) soluciones visitadas recientemente. La idea es explorar a fondo una región del espacio de búsqueda.
  - **Largo Plazo**: Basada en frecuencias. Puede penalizar o premiar (atributos) soluciones muy usadas.
- Diferentes tipos de atributos: Recientes y frecuentes.

- Tipos de estrategias: Intensificar versus Diversificar.

Tabu Search

- **Intensificar** consiste en regresar a buenas regiones ya exploradas para estudiarlas más a fondo. Para ello se favorece la aparición de aquellos atributos asociados a las buenas soluciones encontradas.
- **Diversificar** consiste en visitar nuevas áreas no exploradas del espacio de soluciones. Para ello se modifican las reglas de elección para incorporar a las soluciones atributos que no han sido usado frecuentemente.

Bee Colony Optimization

- Abejas scout abandonan el panal con el objetivo de explorar áreas en la localidad.
- El objetivo es encontrar fuentes de néctar, polen y propóleo.
- Las abejas scouts regresan al panal y reportan sobre las ubicaciones y la calidad de las fuentes encontradas. Se hace a través de un baile.
- Las abejas scouts bailan en un “área de baile” para promover sus fuentes de comida y hacer que los otros miembros las sigan.

Bee Colony Optimization

- Cuando las abejas “devoradoras” abandonan el panal en búsqueda de comida lo hacen siguiendo una de estas abejas scout.
- Cuando la abeja “devoradora” regresa al panal con la carga de néctar, se lo pasa a una abeja “bodeguera”.
- La abeja “devoradora” ahora puede:
  - Puede olvidarse de la ubicación de comida y caer en el receso.
  - Puede continuar explotando el lugar de comida sola.
  - Puede continuar explotando el lugar de comida recrutando nuevas abejas.

Bee Colony Optimization

- La abeja escoge entre una de las tres opciones dependiendo de la calidad, cantidad y distancia al panal.
- Y el proceso se repite iterativamente.
- Es un método constructivo, es decir, construye soluciones desde “scratch” mientras se ejecuta.

Bee Colony Optimization

- Una colonia de B abejas artificial buscan la solución a un problema combinatorio de manera colaborativa.
- Hay dos fases: *forward* and *backward pass*.
- En la "*forward pass*", cada abeja artificial visita NC partes de la solución, creando una solución parcial. Después regresa al panal.
- En el caso de las abejas artificiales, el panal no existe es sólo un elemento de sincronización.

Bee Colony Optimization

- En el panal las abejas intercambian información acerca del estado actual de la búsqueda.
- En cada "*forward pass*" las abejas deben visitar NC componentes.
- Una vez que las abejas han completado la "*forward pass*", comienzan la "*backward pass*".
- En esa pasada, las abejas comparten información acerca de las soluciones parciales.

Bee Colony Optimization

- Teniendo todas las soluciones evaluadas, cada abeja decide si se mantiene fiel a su solución.
- Las abejas con mejores soluciones tienen más chances de promocionar y mantener sus soluciones.
- Las que permanecen en su solución son al mismo tiempo reclutadoras.
- Cuando una solución es abandonada, la abeja se una a de las soluciones promocionadas.

Bee Colony Optimization

- Esa decisión se toma con una probabilidad de manera que las mejores soluciones promocionadas tienen una mayor chance de ser elegidas.
- Dentro de cada “backward pass”, las abejas son divididas entre reclutadoras y no-comisionadas (serán re-asignadas).

Bee Colony Optimization

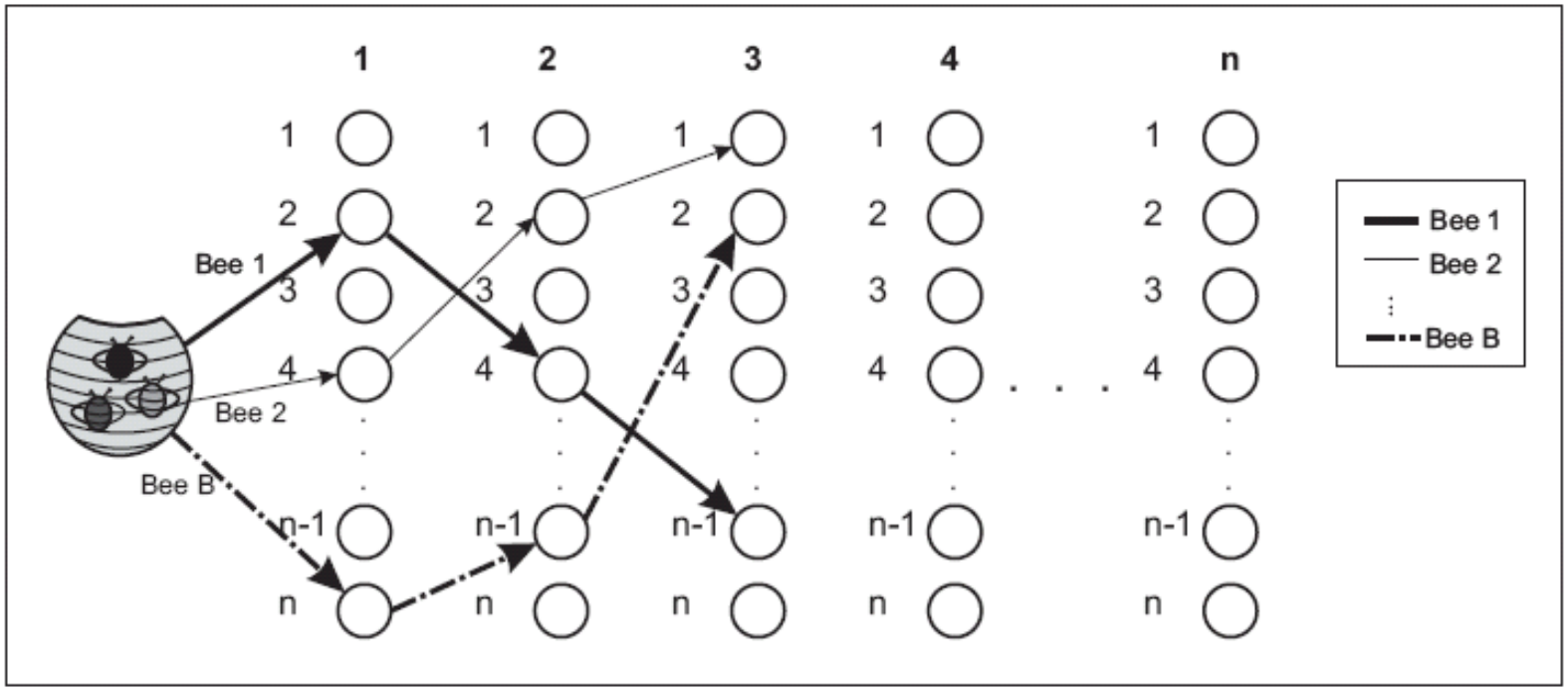


Fig. 1. An illustration of the third forward pass

Bee Colony Optimization

- Por ejemplo, después de evaluar las soluciones parciales la abeja 2 decide unirse a la abeja B.
- Eso implica que ambas abejas “vuelan juntas”, lo que en la práctica significa que se copian las soluciones.
- De ahí cada abeja puede seguir su propio camino.
- En el ejemplo, la abeja 1 sigue su camino.

Bee Colony Optimization

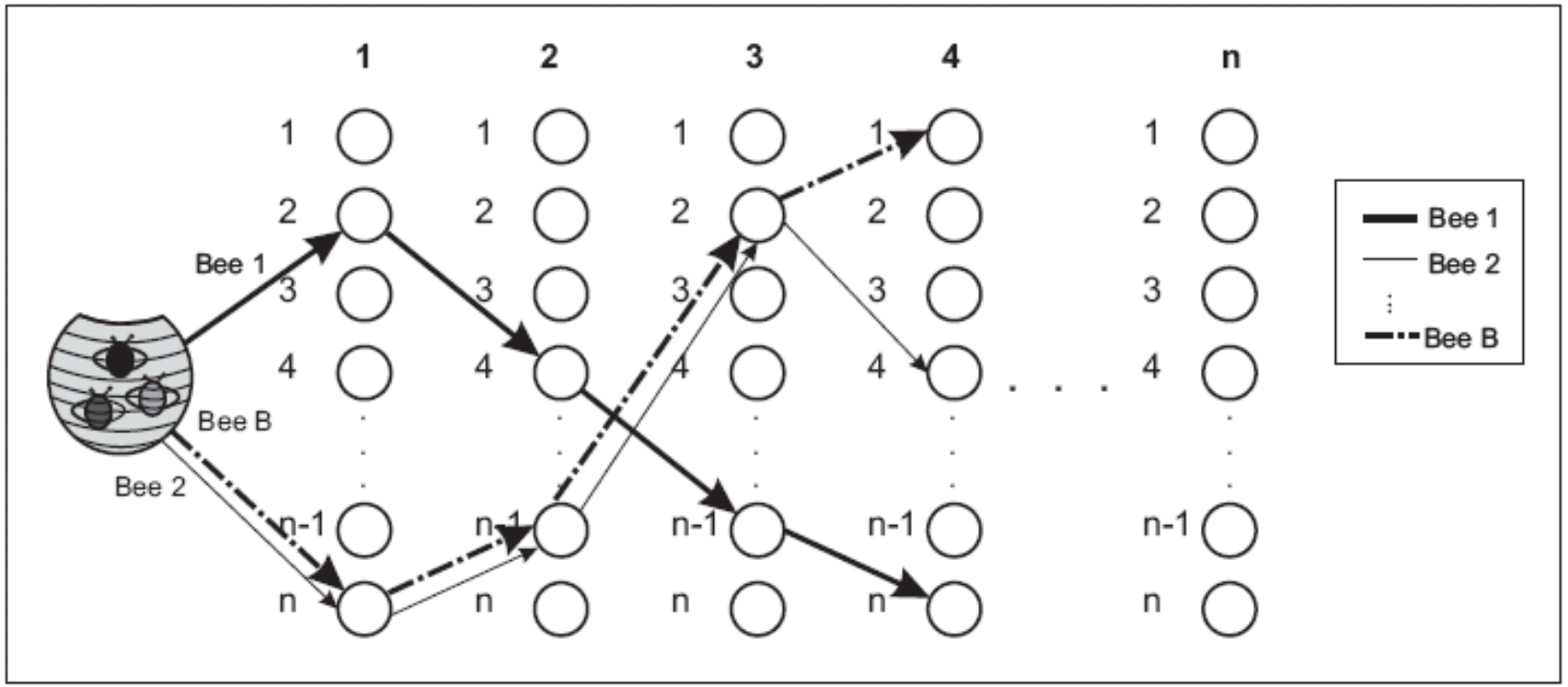


Fig. 3. An example of partial solutions after fourth forward

Bee Colony Optimization

- Las dos fases se alternan hasta que se logra construir todas las soluciones factibles.
- Se actualiza la mejor solución global, se eliminan las B soluciones generadas.
- Hay que diseñar:
  - Como se seleccionan las componentes.
  - Como evaluar y comparar soluciones parciales.
  - Probabilidades: lealtad, reclutamiento, e.g., la ruleta.

Bee Colony Optimization

- B = número de abejas en el panal.
- NC = número de pasos constructivos en un “forwars pass”.
- Al inicio todas las abejas están en el panal.

Bee Colony Optimization

1. Inicialización: una solución vacía es asignada a cada abeja.
2. Para cada abeja:
   a) K=1
   b) Evaluar todos los movimientos constructivos.
   c) Escoger uno utilizando la ruleta.
   d) K++
   e) If k <= NC then ir a (b)
3. Todas las abejas regresan al panal (backward pass).
4. Evaluar las soluciones parciales.
5. Cada abeja decide individualmente si sigue en su solución, se convierte en reclutadora o seguidora.
6. Para las seguidoras, escoger una solución de las recrutadoras.
7. Si las soluciones no están completas, ir al paso 2.
8. Evaluar todas las soluciones y encontrar la mejor.
9. Si no es cumplida la condición de termino ir al paso 2.
10. Mostar la mejor solución.

Bee Colony Optimization

- En general, los pasos 2 y 4 son dependientes del problema.
- **La decisión de lealtad**: La probabilidad de que la b-ésima abeja sea leal, al comienzo de la nueva forward pass, a su solución previamente generada:

$$P_b^{u+1} = e^{-\frac{O\max - Ob}{u}}, b = 1,2,\ldots, B.$$

Bee Colony Optimization

- O_b = El valor normalizado para la solución parcial creada por la abeja b.
- O_{max} = El máximo de todos los valores de las soluciones parciales.
- u=el número de la “forward pass”.
- Entre mejor es la solución, mayor es la probabilidad de que la abeja se mantenga leal.
- Entre más grande la “forward pass”, más grande la influencia de las soluciones parciales ya descubiertas.

Bee Colony Optimization

- Al comienzo las abejas son más bravas para explorar el espacio de búsqueda, a medida que u crece, entonces se vuelven menos bravas.
- El proceso de reclutamiento: La probabilidad de que la solución parcial de la abeja b sea seguida por una abeja ya no-asignada:

$$p_b = \frac{O_b}{\sum_{k=1}^{R} O_k}, b = 1,2,\ldots,R.$$

Bee Colony Optimization

- O_k = el valor normalizado de la función objetivo de la k-ésima solución parcial, y R es el número de reclutadoras.
- Utilizando una generador de números aleatorios, cada abeja se una a una reclutadora.
- En el TSP, las abejas pueden ir agregando ciudades de manera constructivas a cada uno de los tours.
- Se puede calcular la función objetivo de manera parcial.
- Se puede escoger la próxima ciudad con una probabilidad.

Bee Colony Optimization

Table I. TSP benchmark problems: The results obtained by the B

Problem name	No. of nodes	Optimal value	BCO	Relative error (%)	CPU (sec)
Eil51	51	428.87	428.87	0.00	29
Berlin52	52	7544.37	7544.37	0.00	0
St70	70	677.11	677.11	0.00	7
Pr76	76	108159.00	108159.00	0.00	2
Kroa100	100	21285.40	21285.40	0.00	10
Eil101	101	640.21	640.21	0.00	61
Tsp225	225	3859.00	3899.90	1.06	11651
A280	280	2586.77	2608.33	0.83	6270
Pcb442	442	50783.55	51366.04	1.15	4384
Pr1002	1002	259066.60	267340.70	3.19	28101

Particle Swarm Optimization

- Es inspirado por el comportamiento social de los individuos.
  - En este algoritmo se llaman partículas.
- La idea es que estos individuos se mueven en un espacio n-dimensional, donde cada partícula representa una solución potencial al problema.
  - Además, cada partícula puede recordar la mejor posición en la cual ha estado desde el inicio de la búsqueda.
- Todas las partículas comparten su información acerca el espacio de búsqueda.
  - Con este objetivo se guarda una mejor solución global.

Particle Swarm Optimization

- Todas las partículas están buscando el óptimo.
- Dado que todas las partículas se están moviendo tienen asociada una velocidad.
- En cada iteración, cada partícula calcula su velocidad de acuerdo a la siguiente fórmula:

$$V_p(t) = \omega V_p(t-1) + n_1 * rand_1() * (P_p(t-1) - X_p(t-1)) + n_2 * rand_2() * (P_g(t-1) - X_p(t-1))$$

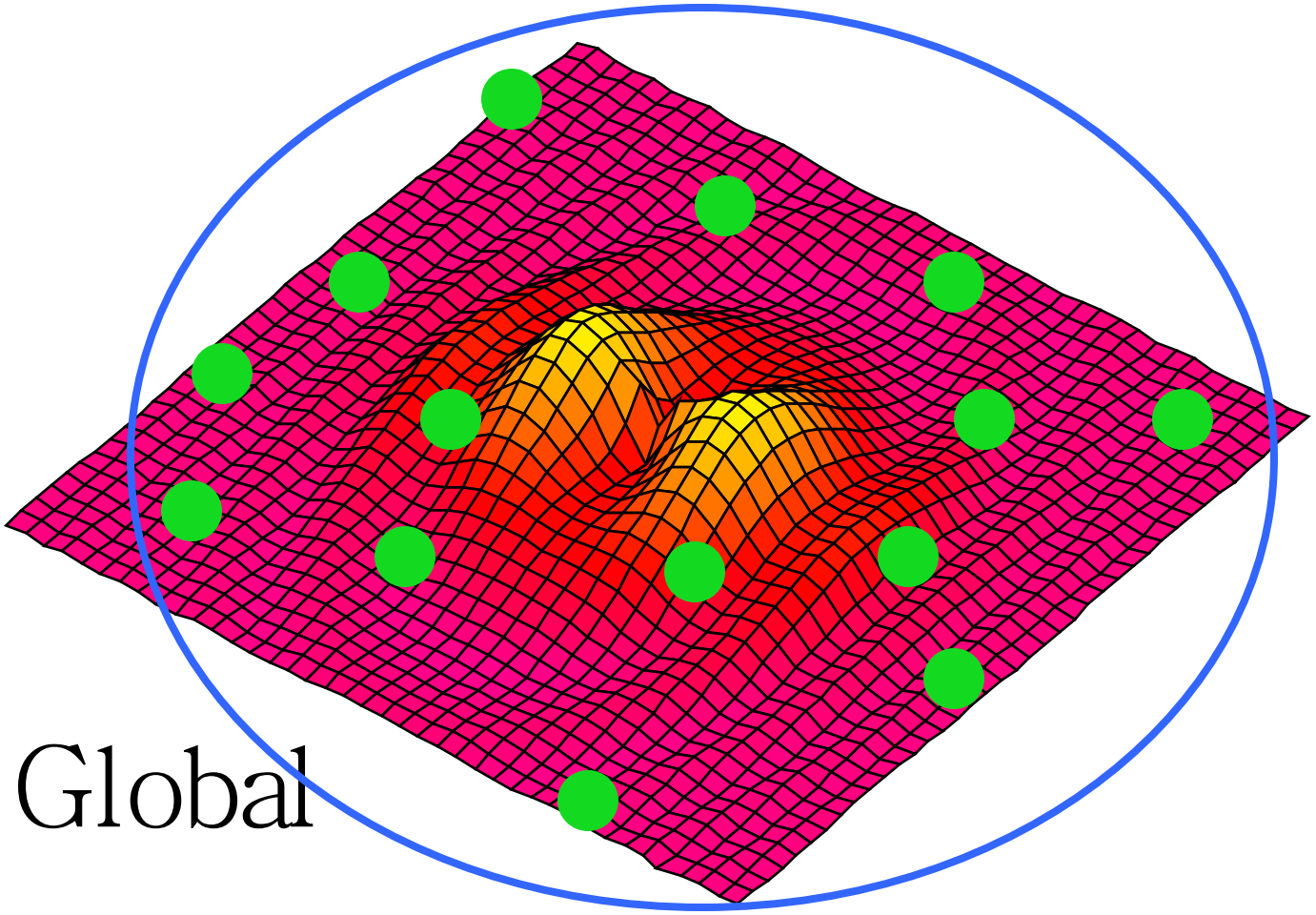
$X_p(t)$ = La posición en la iteración t de la partícula.
$P_p(t)$= La mejor solución que ha alcanzado la partícula hasta la iteración t.
$P_g(t)$= es la mejor solución que han alcanzado entre todas las partículas.
n_1 y n_2 = son pesos que determinan la influencia de P_p y P_g. Son llamados factores de aceleración.

Particle Swarm Optimization

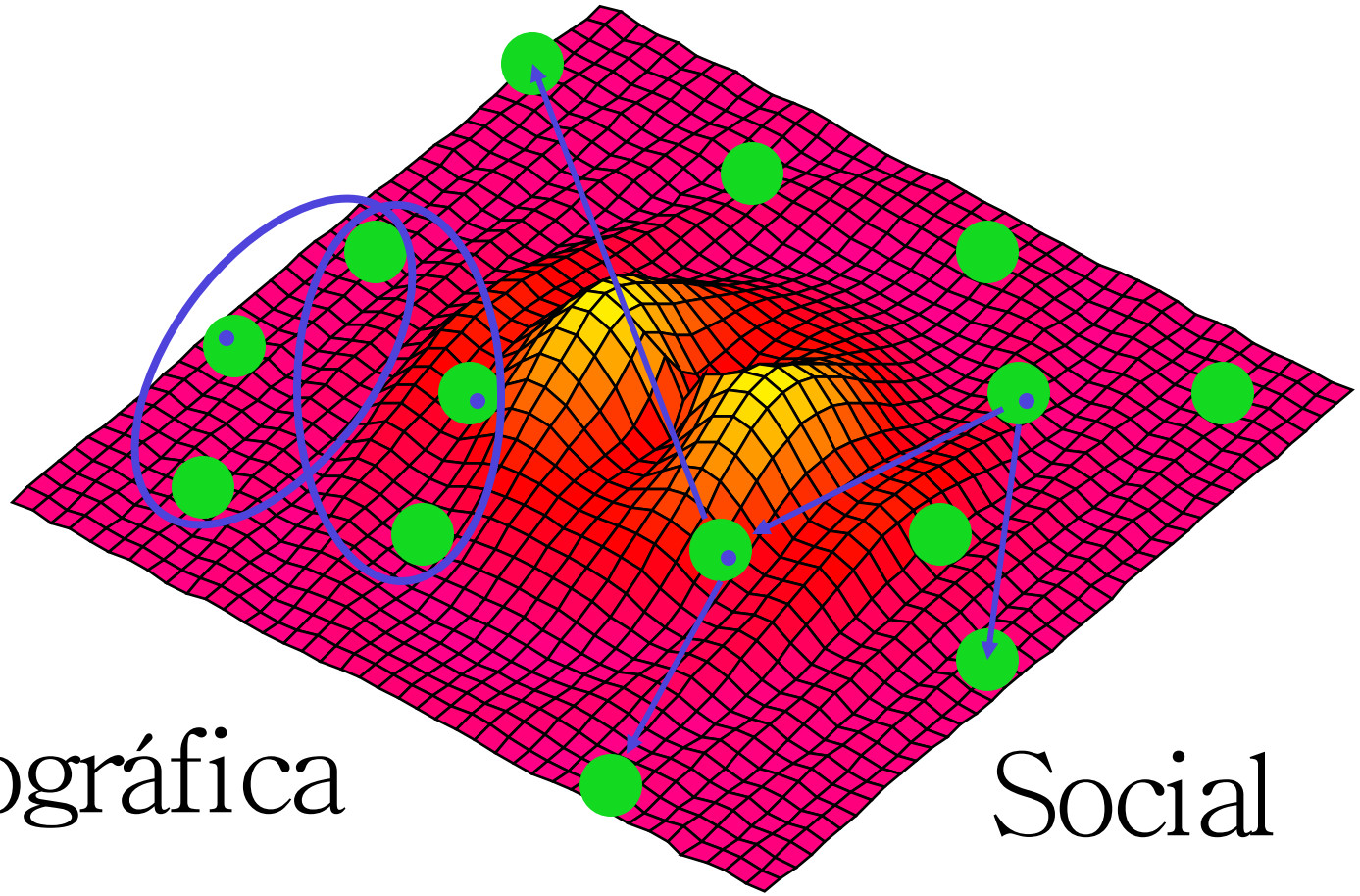
- Después de calcular V_{id}, se obtiene la posición en la nueva iteración:

$$X_p(t) = X_p(t-1) + V_p(t)$$

Particle Swarm Optimization



Particle Swarm Optimization



Geográfica

Social

Particle Swarm Optimization

- En resumen, hay diversas formas de definir la vecindad:
  - **Geográfica**: Sólo toma en cuenta las partículas más cercanas.
  - **Social**: mantiene una lista de vecinas sin importar donde estén. Es una ventaja ya que no es necesario definir una métrica de distancia.
- Resulta evidente que cuando el algoritmo converge, la vecindad social termina siendo también geográfica.
- El tamaño de la vecindad también es un problema. Pero se ha visto que PSO no es muy sensible a este parámetro.

Particle Swarm Optimization

- PSO es una técnica evolutiva, porque comparte ciertos atributos:
  - Tiene un proceso de inicialización donde se construye una población inicial.
    - Con ciertos atributos aleatorios.
  - Construye mejores soluciones en el espacio de búsqueda basado en la generación previa.
  - La nueva generación está basada en la generación previa.

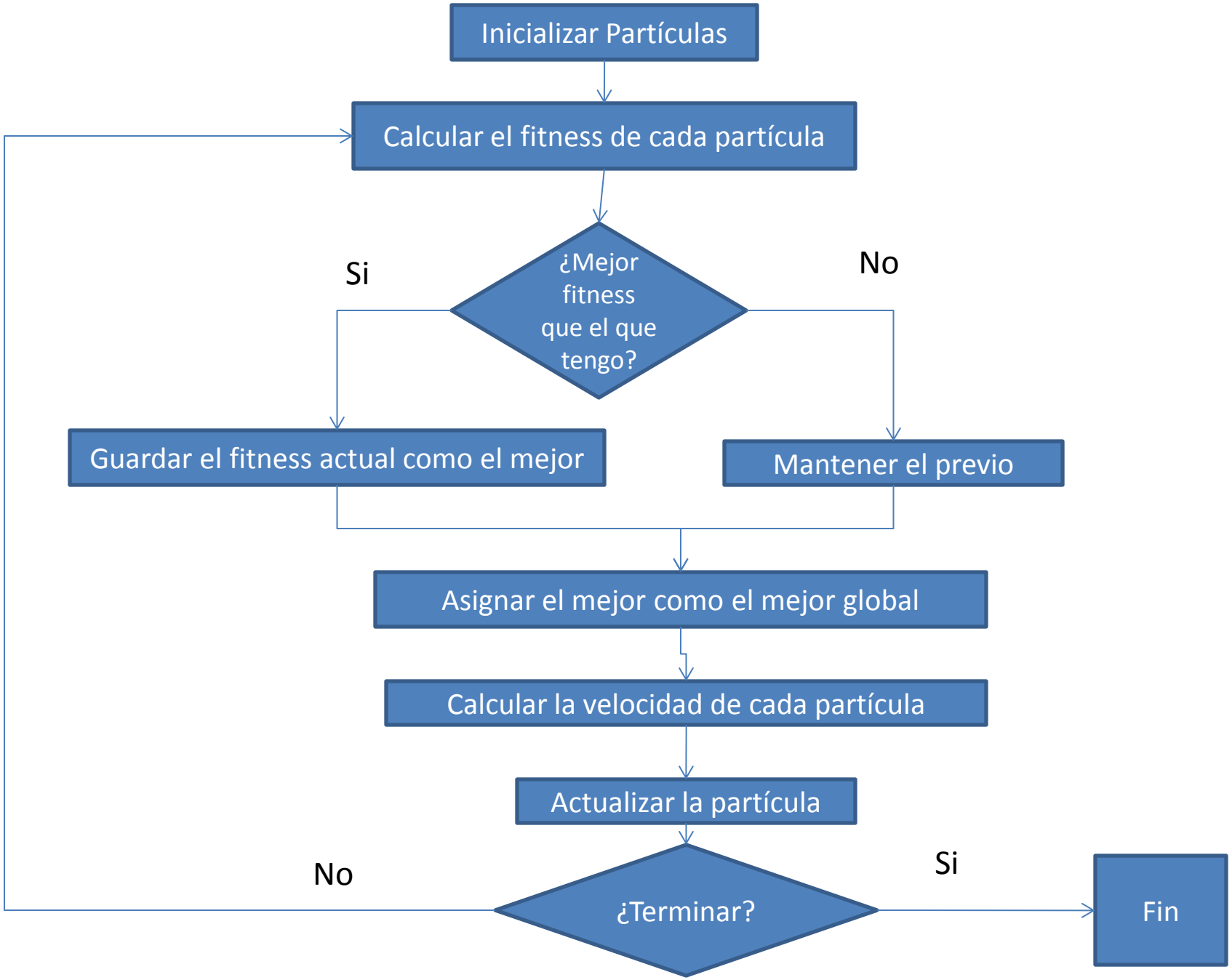
Particle Swarm Optimization

- El movimiento de una partícula está compuesta de tres elecciones posibles:
  - Sigue su propio camino.
  - Regresa o es influenciada por la mejor solución que ha visitado.
  - Se acerca o es influenciado por la mejor solución que ha tenido o tiene alguno de sus vecinos.

Particle Swarm Optimization

```
//1. Inicialización
Para cada Partícula
   Inicializar partícula
Fin
//2. Proceso Iterativo
Hacer
   Para cada Partícula
      Calcular el fitness
      Si este valor es mejor que el mejor personal entonces
                almacenarla como la mejor personal
   Fin

   Escoger la mejor partícula encontrada hasta ahora como la mejor global
   Para cada Partícula
      Calcular su velocidad
      Calcular su posición
   Fin
Mientras no se cumpla un criterio de término
```


Inicializar Partículas
Calcular el fitness de cada partícula
¿Mejor fitness que el que tengo?
Si
No
Guardar el fitness actual como el mejor
Mantener el previo
Asignar el mejor como el mejor global
Calcular la velocidad de cada partícula
Actualizar la partícula
¿Terminar?
No
Si
Fin

Particle Swarm Optimization

- Nótese que las velocidades de cada partícula en cada dimensión están normalmente restringidas a ciertos máximos (V_{max}).
  - Este es un parámetro especificado por el usuario.
- El número de partículas oscila entre 10 y 50.
- Experimentalmente también se muestra que $n_1 + n_2 = 4$ es un buen criterio.
  - Estos factores aseguran la convergencia del algoritmo.
- El factor de inercia ω multiplica la velocidad de la iteración previa.
  - Es normalmente disminuido a lo largo de la ejecución del algoritmo.
  - Crea la tendencia para que la partícula siga la misma dirección que antes.
  - La intención es controlar mejor el balance entre diversificación e intensificación
- Hay que adaptar PSO para ser utilizado en problemas discretos.

Referencias

- Dorigo M., G. Di Caro and L. M. Gambardella. Ant Algorithms for Discrete Optimization. Artificial Life, 5,2, pp. 137-172, 1999.
- Maniezzo V, Gambardella L.M., De Luigi F. Ant Colony Optimization, New Optimization Techniques in Engineering, by Onwubolu, G. C., and B. V. Babu, Springer-Verlag Berlin Heidelberg, 101-117, 2004.
- Gambardella L.M, Dorigo M., Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem, Twelfth International Conference on Machine Learning, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann, 1995, pp. 252-260.
- Gambardella L.M, Dorigo M., Solving Symmetric and Asymmetric TSPs by Ant Colonies, ICEC96, Proceedings of the IEEE Conference on Evolutionary Computation, Nagoya, Japan, May 20-22, 1996

Referencias

- Dorigo M., Gambardella L.M, Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem , IEEE Transactions on Evolutionary Computation Vol. 1,No. 1,pp. 53-66, 1997
- http://www.cs.sunysb.edu/~cse352/GeneticAlgorithms.pdf
- http://es.wikipedia.org/wiki/Metaheur%C3%ADstica
- **"*Particle Swarm Optimization for Traveling Salesman PROBLEM*", K. Wang, L. Huang, C. Zhou, P. Wang**
- **"Particle Swarm Optimization Algorithm for the**
- Elizabeth F. G. Goldbarg, Marco C. Goldbarg and Givanaldo R. de Souza (2008). Particle Swarm Optimization Algorithm for the Traveling Salesman Problem, Traveling Salesman Problem, Federico Greco (Ed.), ISBN: 978-953-7619-10-7, InTech, Available from: http://www.intechopen.com/books/traveling_salesman_problem/particle_swarm_optimization_algorithm_for_the_traveling_salesman_problem