**Guía 7 – 2023 – Anotaciones**

**Colonia de hormigas y enjambres de partículas**

**EJERCICIO 1**

<https://drive.google.com/drive/folders/1U0tUQK7EB05OsQpdo2_qdjOlkuqAzrUH>

Se pide implementar el algoritmo de optimización con enjambre de partículas y utilizarlo también para encontrar el mínimo global de las funciones del ejercicio 1 de la guía 6. Luego se pide comparar los resultados de éste método con los del algoritmo genético del ejercicio 1, en términos de las soluciones encontradas y la velocidad de convergencia.

**Repaso del método de optimización por enjambre de partículas**

(En la teoría están desarrollados más en profundidad).

-Son métodos de búsqueda inspirados en el comportamiento de bandadas de pájaros. Se basa en el comportamiento de los individuos de imitar el éxito de otros individuos además de considerar su propia experiencia. A partir de esto se logra que, mediante reglas individuales y simples se logren comportamientos sociales complejos.

Entonces en un enjambre cada partícula consiste de:

-Posición actual **x**i(t), donde “i” indica la partícula del enjambre, “t” es el instante de tiempo, y el vector “**x**” en cada dimensión tiene una variable del problema, es decir, en este caso no hay una codificación como en los algoritmos genéticos, sino que las variables están en el mismo dominio del problema.

-Velocidad actual **v**i(t), también es un vector de las mismas dimensiones.

-Mejor posición histórica yi(t), es decir, la posición en la que obtuvo el mejor fitness hasta ese momento.

-Mejor posición histórica global o de su vecindario ^yi(t), es información sobre su enjambre, la mejor posición obtenida en todo el enjambre hasta ese momento. (El ^ es un sombrerito en la “y”).

**Nota**: Cuando la mejor posición global no usa el subíndice “i” sino que es solo ^y, representa la mejor posición histórica global para todo el enjambre, mientras que si es ^yi no va a ser la misma para todo el enjambre, sino que va a depender de la vecindad en la que se encuentre la partícula, es decir, la mejor posición histórica de la vecindad de esa partícula.

**Pseudocódigo del algoritmo de optimización para enjambres de partículas**

*En 2023 el profe mostró otra imagen con el pseudocódigo, que es la del Algoritmo 3 que vimos en teoría (casi al final del Word de la teoría), pero creo que es lo mismo que explico abajo.*

Debajo vemos la imagen del pseudocódigo, en el cual, el primer paso es inicializar los vectores de posición de cada partícula en el enjambre, esto lo hacemos de forma aleatoria y con distribución uniforme. La aleatoriedad en las posiciones iniciales permite que el algoritmo explore diferentes regiones del espacio de búsqueda en busca de la solución óptima.

Luego comienza el bucle, donde para cada partícula “i” desde 1 hasta ns que sería el tamaño del enjambre, en el bucle primero vamos a calcular la función objetivo evaluada en la posición actual de la partícula f(xi), y la vamos a comparar con la función objetivo evaluada en la mejor posición histórica de esa partícula f(yi). Si la función objetivo es mejor que la mejor posición obtenida para esa partícula “i” hasta el momento, vamos a actualizar esa mejor posición histórica de la partícula, es decir, yi = xi. Como estamos actualizando solo si la función objetivo en la posición actual es menor, significa que estamos minimizando la función objetivo, a diferencia del algoritmo genético donde maximizamos el fitness.

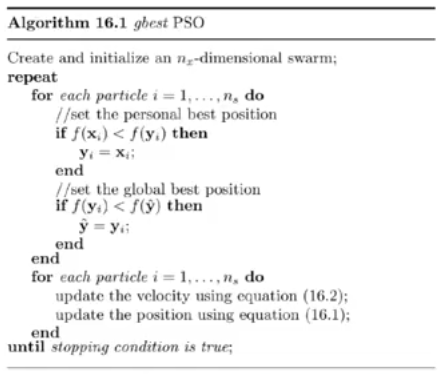
Para la primera iteración, la mejor posición histórica va a ser la misma que la posición actual, por lo tanto, ese if pasa de largo la primera vez.

Luego en el segundo if vamos a comparar la función objetivo evaluada en la mejor posición historia de la partícula f(yi) contra la función objetivo evaluada en la mejor posición global f(^y), es decir, la mejor posición histórica en el enjambre, y si fuera mejor, actualizamos la mejor posición del enjambre con esa mejor posición de la partícula: ^y = yi.

Luego, en el segundo for, para cada partícula vamos a actualizar la velocidad y después actualizar el vector de posiciones.

Esto lo vamos a iterar hasta que se cumpla la condición de corte, en este caso sería el pseudocódigo para la variante global del algoritmo, donde se considera esa mejor posición global ^y de todo el enjambre. La diferencia con la variante local es que esa mejor posición en vez de considerarla de todo el enjambre vamos a tener una mejor posición histórica para cada vecindad, es decir, cada partícula va a tener una vecindad y solo va a conocer la mejor posición dentro de esa vecindad y no dentro de todo el enjambre.

*(****Pensar*** *que diferencias o ventajas/desventajas tendría utilizando las dos variantes: mejor posición global y mejor posición local, por si se pregunta).*



Lo que nos faltaba definir en el pseudocódigo es cómo actualizar la posición de la partícula y cómo actualizar la velocidad, como se muestra en la imagen de abajo.

En el caso de la actualización del vector de posición, la posición de la partícula “i" en el instante t+1: xi(t+1) va a ser la posición actual de la partícula más el vector de velocidad.

Mientras que, para actualizar el vector de velocidad (global best), es el corazón de este algoritmo. En la imagen se puede notar un pequeño cambio de notación, donde estamos descomponiendo el vector de velocidad vi en sus distintas componentes, por eso aparece el índice “j” en vij(t+1), donde “j” representa cada una de las dimensiones del vector **v**i, que son las mismas dimensiones que las del vector “**x**”.

Entonces, para actualizar la velocidad de la partícula “i” en la dimensión “j” para el instante t+1, es decir, vij(t+1), vamos a calcularlo como se muestra en la imagen, donde sumamos la velocidad actual de la partícula en esa dimensión vij(t), y tenemos luego un término que corresponde a la experiencia personal de la partícula (todo el término con c1r1 y los corchetes), y un segundo término que corresponde a la experiencia del enjambre (la suma del c2r2 y el corchete). Se puede ver que en el primer término consideramos la mejor posición histórica yij de la partícula “i”, y en el segundo término se considera la mejor posición histórica del enjambre ^yj, por lo tanto, *ese primer término lo que hace es tratar de llevar esa partícula hacia la dirección donde obtuvo su mejor desempeño, mientras que el otro término trata de llevar la partícula hacia la mejor posición global.*

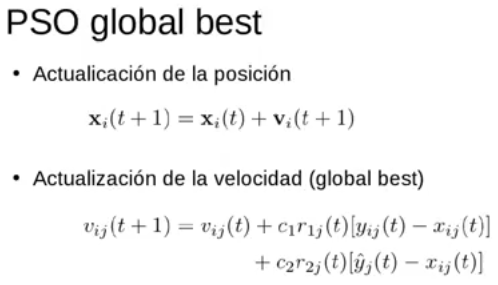
*Tenemos un vector* ***r****1j y un vector* ***r****2j*, que también tienen el subíndice “j”, por lo tanto, son vectores con la misma dimensión que el vector velocidad (y que el vector **x**), y son vectores aleatorios que se generan con distribución uniforme entre 0 y 1, y de esa manera se agrega una componente estocástica (aleatoria) al algoritmo, similar a lo que sería el operador de mutación en los algoritmos genéticos. Es importante notar que los vectores **r**1 y **r**2 al tener diferentes valores aleatorios en cada una de sus componentes, permiten que la influencia de la mejor posición de la partícula y la mejor posición global tengan diferente peso en cada una de las dimensiones, lo cual permite a la partícula hacer una exploración más completa del espacio de búsqueda, le permite moverse con mayor libertad. Además esos vectores dependen del tiempo “t”.

*Y las dos constantes “c1” y “c2” son dos constantes del algoritmo que nos permiten controlar cuánta importancia le damos a la experiencia personal de la partícula y cuánta a la experiencia del enjambre, en el cálculo de la velocidad.*

Si c1 es mayor que c2, le estaríamos dando mayor peso a la componente que representa la experiencia personal de la partícula, por lo tanto, en ese caso el algoritmo hace una mayor exploración del espacio de búsqueda, mientras que si c2 es mayor que c1, estaríamos favoreciendo a la convergencia del algoritmo pero a su vez limitando un poco la exploración del espacio de búsqueda.

Entonces lo ideal es hacer que esas dos constantes (c1 y c2) varíen a lo largo de las iteraciones, de esa forma se puede hacer que c1 sea grande al principio para que en las primeras iteraciones se haga una buena exploración del espacio de búsqueda y luego se vaya reduciendo, mientras que para c2 hacemos que sea pequeña al comienzo y luego que vaya creciendo a lo largo de las iteraciones, para que en las últimas iteraciones se le vaya dando más importancia a la componente social del enjambre, y de esa forma se favorezca a la convergencia del algoritmo.

En general, para c1 y c2 se suelen utilizar valores entre 0.5 y 2.5.



***Recordar*** *bien esa fórmula y que significa cada cosa porque tal vez se puede preguntar en el parcial*

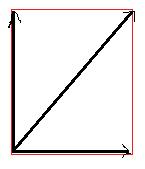
En nuestro código, probamos distintos valores para c1 y c2 para observar como varía la búsqueda o movimientos que realizan las partículas, observando eso que mencionamos arriba donde por ejemplo si c1 es mayor que c2 hace una mayor exploración del espacio de búsqueda. Se ve en la gráfica si por ejemplo le damos un valor más alto a c1 y a c2 la dejamos con un valor bajo, así en las primeras iteraciones las partículas se mueven mucho más y luego se van controlando porque va bajando el valor de c1 y subiendo el de c2 en nuestro algoritmo.

Otras cosas que mencionó el profe en la clase sobre estos “c” y “r”:

-Inicialmente es bueno que cada partícula busque por su cuenta, y que luego se vayan juntando, entonces eso también sirven las constantes “c1” y “c2” del algoritmo.

-Los “r” también añaden esas pequeñas variaciones, entonces no me muevo tal cual en la dirección que me indica sino que me muevo con esas variaciones, y eso añade algo de aleatoriedad para hacer una exploración más completa del espacio de búsqueda como dijimos.

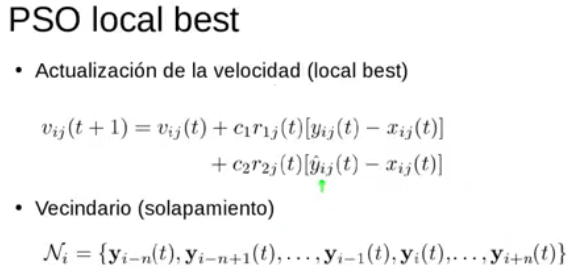
Si tengo algo como en la imagen de abajo, donde la dirección original para moverme es esa negra en diagonal, si tengo un caso extremo de r = [1 0] me voy a mover en la dirección horizontal, mientras que si tengo el otro caso extremo r = [0 1] me voy a mover en la dirección vertical, y con cualquier otra variación me voy a mover dentro de ese rectángulo rojo, es decir que no es que va a buscar en cualquier lado del espacio, sino que mantiene más o menos la dirección original pero la varía un poco de forma aleatoria según los valores de r1 y r2, así que se mantiene buscando dentro de ese rectángulo.



Ya vimos como actualizar la velocidad para el caso de la versión global del algoritmo.

Para la **versión local** la actualización es similar, lo que cambia es que la componente social (marcada con una flecha en la imagen de abajo), ya no vamos a tener en cuenta la mejor posición de todo el enjambre, sino la mejor posición histórica de la **vecindad** de esa partícula. Por eso ahora aparece el índice “i” en la mejor posición histórica social, es decir, ^yij porque ya no va a ser la misma para todo el enjambre, sino que va a depender de la vecindad en la que se encuentre la partícula.

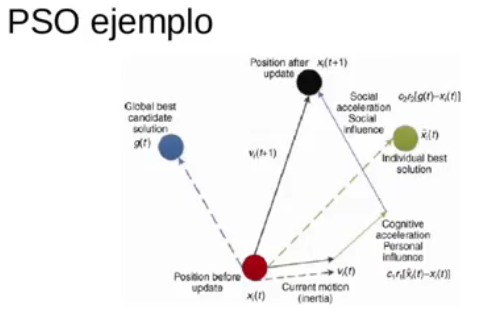
Entonces podemos definir una vecindad para la partícula “Ni”, por ejemplo, una vecindad lineal de tamaño “n”, entonces esa vecindad va a incluir las partículas desde i-n hasta i+n, y entre esas partículas van a compartir la información social, es decir, la mejor posición histórica de la vecindad.



***Recordar*** *y entender estas fórmulas también y todas las que vemos, por si se pregunta en el parcial*

En la imagen de abajo vemos un ejemplo que ilustra cómo se actualiza la posición en la partícula, teniendo en cuenta la información de la mejor posición en el enjambre y la experiencia individual de la partícula. Tenemos la posición actual x(t) roja, la mejor posición del enjambre es la azul, y la mejor posición individual de la partícula es la verde.

Entonces según cuánto peso se les dé a cada componente, se actualiza el vector velocidad v(t+1)y luego se actualiza la posición de la partícula x(t+1), que sería la negra en la imagen, en este caso más o menos entre medio de la mejor posición global y la mejor posición personal, por lo tanto, se estaría explorando una nueva zona del espacio de búsqueda que no había sido explorada.



**Consideraciones finales:**

-En el caso de la versión global del algoritmo, la mayor interconectividad le permite una convergencia más rápida, pero en el caso del algoritmo local tener vecindarios aislados le permite conservar mayor diversidad, y esto lo hace menos susceptible a caer en mínimos locales.

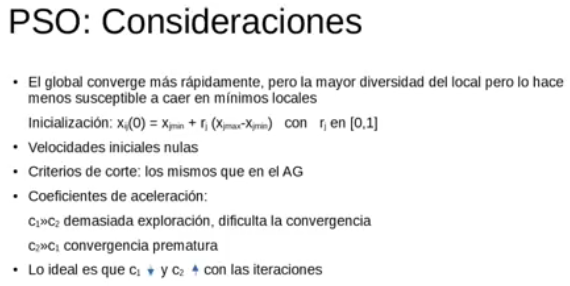
-Para la inicialización, si conocemos los valores máximos y mínimos para cada dimensión del problema, podemos aprovecharlo para inicializar las posiciones de esa manera que se muestra en la imagen de abajo, generando un “r” aleatorio entre 0 y 1 con distribución uniforme y de esa manera nos aseguramos que inicializamos las posiciones dentro de los rangos adecuados.

-Las velocidades iniciales se setean como nulas.

-Como criterios de corte podemos tener los mismos que mencionamos para algoritmos genéticos.

-Para los coeficientes de aceleración, las constantes c1 y c2, si c1 > c2, le damos más importancia a la componente individual que a la social, entonces se realiza una mayor exploración pero se dificulta la convergencia, mientras que si c2 > c1 le damos mayor importancia a la componente social, entonces tenemos riesgo de una convergencia prematura.

Lo ideal dijimos que es que c1 sea grande comparada con c2 al principio, en las primeras iteraciones y luego vaya decreciendo, mientras que c2 sea pequeña al principio y luego vaya incrementándose para que el algoritmo tienda a converger en las últimas iteraciones pero pueda realizar una buena exploración del espacio en las primeras iteraciones.



**EJERCICIO 2**

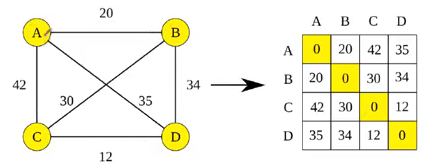
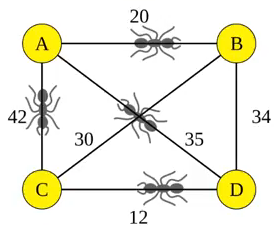
<https://drive.google.com/drive/folders/1U0tUQK7EB05OsQpdo2_qdjOlkuqAzrUH>

En la imagen vemos un ejemplo para este problema del agente viajero, donde tenemos cuatro ciudades (A, B, C, D), conectadas entre sí. El problema consiste en partir desde una ciudad y hacer un recorrido que nos permita pasar por las otras tres ciudades y volver a la de origen, sin repetir ninguna ciudad. Por ejemplo, un recorrido que sea: A – B – D – C – A.

Es un problema simétrico, ya que viajar a “A” a “B” o de “B” a “A” tiene la misma longitud.

También se puede representar con una matriz de conexión como se muestra en la imagen, donde la intersección entre una fila y una columna nos dice la distancia que hay para viajar entre dos ciudades.

La diagonal en esa matriz podemos ver que son todos 0, ya que viajar de una ciudad a la misma no tiene distancia. Y es simétrico como dijimos antes porque en este caso viajar de A hacia B o de B hacia A es lo mismo.

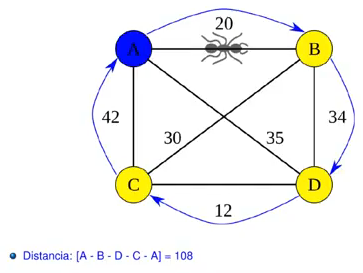
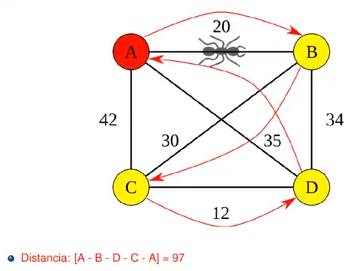
En este contexto, aplicar el problema de hormigas consistirá en poner cada hormiga en un nodo, podría ser todas en el mismo o en nodos diferentes, y permitir que cada hormiga haga un recorrido eligiendo las ciudades y volviendo a la de origen y proponga una solución.

Algo interesante para este problema es que *no importa cuál sea el nodo de origen para comenzar el camino, la solución debería ser la misma, ya que es un recorrido cerrado*. Por ejemplo, si hacemos el recorrido A – B – D – C – A, sería equivalente al recorrido D – C – A – B – D.

Entonces la idea es que cada hormiga parta de un nodo y proponga un recorrido teniendo en cuenta las distancias y matriz de feromonas que veremos luego. En la imagen de abajo a la izquierda tenemos un ejemplo donde la hormiga parte del nodo A y hace el recorrido A – B – C – D – A y el recorrido tiene una distancia de 108 unidades.

Otro recorrido sería el de la imagen de la derecha, donde el recorrido es diferente y la distancia total también, ahora es menor.

*La idea es que las hormigas vayan proponiendo recorridos hasta encontrar el de menor longitud*.

**Repaso del algoritmo de sistema de hormigas**.

Algo que anote de la clase de práctica: el profe mostró y explicó el “Algoritmo 2: sistema de hormigas (AS)” que vimos en la teoría.

Ver bien qué términos o variables se usan en la fórmula de la probabilidad “p” que se ve en 3.1.2 de la imagen de abajo (ver lo explicado en el Word de teoría): sigma es la cantidad de feromonas, y ƞ es el costo de hacer ese paso, que en nuestro caso es distancia del camino pero podría ser otra cosa como costo económico o cualquier cosa, y en el denominador una constante de normalización para que la sumatoria me de 1. Y los exponenciales alpha y beta.

Una vez que tengo esas probabilidades yo puedo hacer algo como en la ruleta para elegir cuál es el siguiente paso que va a dar la hormiga, tirando un valor al azar y eligiendo según las probabilidades.

Se comienza inicializando la matriz de feromonas.

La matriz de feromonas es una estructura donde las hormigas van a ir guardando la información de las soluciones que encuentran, para compartirlas con las demás hormigas. Para inicializar esa matriz se sugiere utilizar valores pequeños entre 0 y un valor cercano a 0, extraídos a partir de una distribución uniforme.

En la práctica, también se puede usar una inicialización donde todos los valores sean iguales a 1, total a lo largo del algoritmo ese valor se irá actualizando y no habrá problemas.

Algo interesante a tener en cuenta es que, la matriz de distancia para representar esa ciudad del ejemplo, será de 4x4, ahora bien, la matriz de feromonas también será de 4x4 en este ejemplo, ya que en cada elemento guardamos la cantidad de feromonas que deja la hormiga cuando pasa de una ciudad a otra.

Una vez inicializada la matriz de feromonas, ubicamos “N” hormigas en el nodo de origen. Podríamos poner todas en el mismo nodo o ubicarlas en distintos nodos, total las soluciones que se encuentren serán las mismas al ser un recorrido cerrado. Lo único que si elegimos distintos nodos como nodo inicial, hay que hacer luego un alineamiento para verificar que las soluciones sean iguales (después lo vemos).

Después se comienza con el proceso de búsqueda iterativo. Cada iteración va a consistir en lo siguiente:

-Primero, para cada hormiga definimos una lista vacía donde vamos a almacenar las ciudades en el orden en que se van recorriendo, y el primer elemento de la lista será el nodo o ciudad inicial del que parte la hormiga.

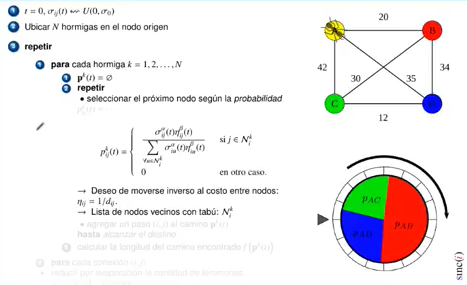
-Luego de forma iterativa, para cada hormiga y dado el nodo en el que estamos parados, vamos a elegir cuál es la siguiente ciudad o nodo a la que nos vamos a mover, con una probabilidad que viene dada por la ecuación de la imagen. En el numerador aparece el valor de feromonas que tenemos almacenado para movernos de la ciudad “i” a la ciudad “j”, que sería σij, por ejemplo, para movernos de A hacia B, y también está el valor “ƞ” (eta) que se calcula como la inversa de la distancia entre las dos ciudades. Ese valor se podría tener almacenado previamente, lo que sí hay que tener cuidado con los elementos de la diagonal, porque las distancias de un nodo consigo mismo son 0, así que habría divisiones por 0 que debemos evitar (se puede poner todos 1 en la diagonal, hacer esas divisiones y luego poner todos 0 en la diagonal).

Mientras que en el denominador tendremos una sumatoria de todos los posibles valores de transición, que serían los productos de “σ” y “ƞ” que se usaban en el numerador, entonces se va a normalizar esos productos del numerador dividiendo por la sumatoria de todos los productos de las transiciones.

Se debe tener en cuenta que en la sumatoria interviene ese parámetro “N” que va a ser una lista de vecinos con tabú. ¿Qué significa eso? Supongamos que del nodo B nos movemos al nodo A, ahora desde A debemos elegir a qué nodo nos vamos a mover, que será C, D o B, pero como venimos de B, esa lista N de posibles nodos para movernos lo que hace es quitar B de las posibilidades para movernos.

-Entonces vamos a calcular una probabilidad “p” de esa manera y luego hacemos lo mismo que en el algoritmo genético, vamos a normalizar esas probabilidades y aplicar una especie de método de ruleta (imagen de la derecha) para elegir a qué ciudad nos vamos a mover. En ese ejemplo, movernos de A hacia B tiene más probabilidad que movernos de A hacia C o hacia D.

Entonces, normalizamos las probabilidades “p” calculadas, tiramos un número al azar entre 0 y 1, y dependiendo de la proporción que represente esa probabilidad en la ruleta nos da la siguiente ciudad donde nos vamos a mover.



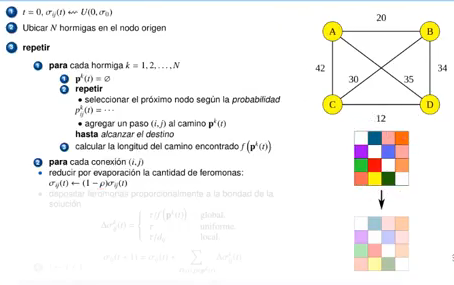
-Una vez elegida la siguiente ciudad, vamos a agregar esa ciudad al camino que estamos construyendo y vamos a repetir esto hasta que no tengamos más ciudades, es decir, hasta que todas las ciudades disponibles formen parte del camino.

Llegados a ese punto, recordar que si partimos de la ciudad A, deberíamos agregar al final de la lista la ciudad A nuevamente, para volver al nodo de origen.

-Luego calculamos la longitud del camino encontrado sumando todas las distancias entre las transiciones de ciudades que fuimos realizando y que forman la solución.

-Una vez completada la búsqueda con cada hormiga, realizamos el proceso de evaporación de feromonas multiplicando la matriz de feromonas por un valor que es la tasa de evaporación. La idea es que las feromonas almacenadas en cada una de las transiciones o caminos, pierda una pequeña proporción debido a la evaporación y la hormiga pierda parte de la información de los caminos que son menos utilizados.

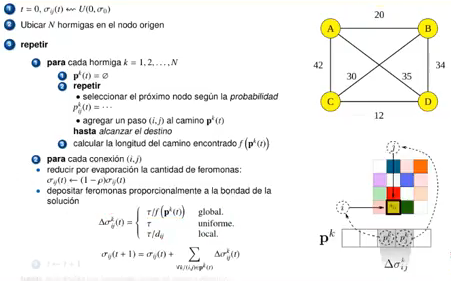
Si bien en la imagen esta expresado como una operación elemento a elemento, el valor “*p*” es constante, se aplica a todos los elementos de la matriz, entonces se puede hacer una operación matricial (multiplicar la matriz de feromonas por la tasa de evaporación).



-Una vez que se tiene eso listo, se hace el depósito de feromonas teniendo en cuenta las soluciones encontradas. Para eso hace falta calcular cuánta feromona vamos a depositar en cada camino que fuimos recorriendo, por ejemplo, para la transición de A hacia B decidir cuánta feromona vamos a dejar. Para eso, como se ve en la imagen de abajo vamos a considerar tres métodos: el método uniforme supone depositar una cantidad constante τ en cada transición, el método global propone dejar una fracción de ese total τ de feromonas, que va a depender de la longitud total del camino, es decir, divide esa cantidad total de feromonas por la longitud total del camino. Cuanto más corto sea el camino más feromonas vamos a dejar, pero en todas las transiciones va a dejar la misma cantidad. Mientras que en el método local, la cantidad de feromonas τ se divide por la distancia que hay para moverse de la ciudad “i” a la “j” (dij), es decir, esa τ que vamos a depositar en cada transición va a ser distinta, ya que las distancias entre nodos o ciudades son distintas.

De esa forma vamos a calcular cuánta cantidad de feromona Δσij vamos a dejar en una transición.

-Teniendo en cuenta ese delta de feromonas, vamos a actualizar la matriz de feromonas recorriendo todas las transiciones que realizaron cada una de las hormigas y, de acuerdo al método de depósito de feromonas que elegimos, ir sumándole al valor de feromonas que tenemos en la matriz, el valor Δσij que calculamos antes de acuerdo al método.



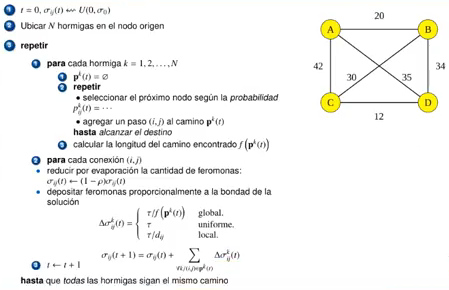
-Una vez hecho esto, comenzamos una nueva iteración y se repite hasta alcanzar un criterio de corte.

-En este punto, vamos a hacer dos aclaraciones.

La primera es que, como toda meta heurística hace falta elegir un número máximo de iteraciones, para evitar que el algoritmo se ejecute de forma indefinida. Además, vamos a elegir un criterio específico para este problema, y es que todas las hormigas sigan el mismo camino, y acá es importante tener en cuenta si todas las hormigas parten del mismo nodo inicial o no. Si elegimos que todas parten del mismo nodo inicial directamente podemos comparar las secuencias que obtuvimos con cada hormiga y deberían ser iguales. Mientras que, si no partimos con todas del mismo nodo, se deben alinear las secuencias para determinar si efectivamente son las mismas soluciones.

No era para este caso, pero el profe había mencionado en la clase que puede que no todas las hormigas se alineen en el mismo camino, depende de la dificultad del problema.

(Lo dejo anotado para recordar nada más).

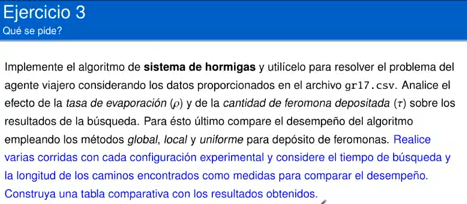


**Otras sugerencias**:

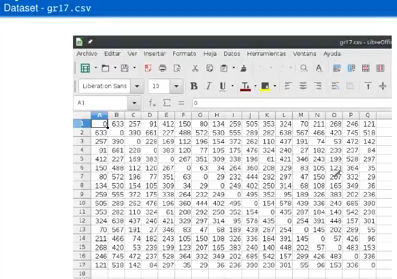
-No nos quedemos con una sola iteración donde se cumpla que todas las hormigas siguen el mismo camino, ya que podría darse aleatoriamente que en una iteración se alinean todas las hormigas y en la siguiente no, entonces conviene tener un contador y, por ejemplo, si todas las hormigas se alinean durante cinco iteraciones consecutivas ahí sí detener el algoritmo.

-Conviene incluir en el algoritmo una hormiga elite, es decir, ir almacenando en cada iteración la mejor solución encontrada hasta el momento, es decir, el mejor recorrido (que tiene la menor longitud), y actualizándolo a lo largo de las iteraciones. Y al final devolvemos la mejor solución.

**Volviendo a lo que se pide en el ejercicio**



Respecto al archivo de datos, tendrá la estructura que se muestra en la imagen, similar a la que vimos en el ejemplo, donde podemos ver que en la diagonal tendrá todos valores 0 (distancias de una ciudad a ella misma), y es simétrico, ya que viajar de A hacia B tiene la misma distancia que viajar de B hacia A.

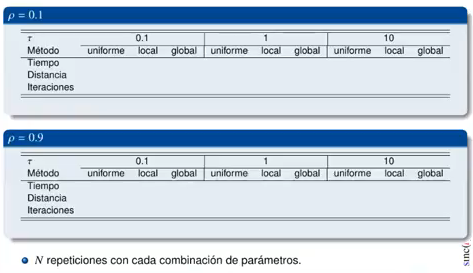


Respecto a lo que se pide, la idea es que creemos tablas similares a las que se muestran abajo.

Eligiendo un valor de tasa de evaporación “*p*”, la idea es que probemos distintos valores de “τ”, que era la cantidad de feromonas que dejamos en el depósito (por ejemplo, 0.1, 1 y 10), y con esos valores y para cada método de depósito de feromonas que vimos, realicemos “N” recorridos.

Por ejemplo, con *p* = 0.1, τ = 0.1 y el método de depósito de feromonas uniforme, repetir N = 10 veces la corrida y con esas 10 soluciones calcular el tiempo promedio que llevo encontrar la solución, la distancia promedio de la solución encontrada y el número de iteraciones promedio utilizado. Y lo mismo para las demás combinaciones de parámetros.

Luego empleando otra tasa de evaporación “*p*” realizamos lo mismo y comparamos cuál es el efecto de esos parámetros sobre la calidad de las soluciones encontradas.



Por lo que pudimos observar probando el algoritmo con distintos valores para la tasa de evaporación, con 0.1 le lleva muchas más iteraciones que por ejemplo con 0.5 o 0.9, es decir, al aumentar la tasa de evaporación mejora la rapidez.

Y respecto a la solución obtenida como mejor camino, se llegó a la misma solución aunque se varíe la tasa de evaporación, algunas veces un poquito mayor distancia pero en general casi siempre una distancia de 2187 o cerca de 2200.