FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

NOMBRE DE LA ASIGNATURA: *INTELIGENCIA ARTIFICIAL*
NOMBRE DEL PROFESOR: *RAMIREZ ORTEGON JUSTINO*
NOMBRE DEL ALUMNO: *ALBA ARCOS CHRISTOPHER GREGORIO*
TEMA: *ALGORITMOS HEURÍSTICOS*
PRÁCTICA NÚM. [**2**]

OBJETIVO:
Implementar la solución al problema del agente viajero el método de Simulated Annealing, usando una interfaz gráfica para visualizar el proceso de solución, el estado inicial y final de la ruta y puntos de visita además de una que se grafique la evolución de la distancia del recorrido.

RESUMEN:
Dado los parámetros iniciales del número de ciudades, número máximo de iteraciones, temperatura y el factor de enfriamiento se calculará la ruta más optima encontrada según el método de Simulated Annealing, pudiendo visualizar las rutas trazadas entre las ciudades, las cuales se muestran en pantalla y son puntos aleatorios.

MARCO TEÓRICO:
El problema del agente viajero Se tiene un número de nodos (ciudades, localidades, tiendas, empresas, etc.) que deben ser visitados por una entidad (persona, agente viajero, automotor, avión, autobús, etc.), sin visitar 2 veces el mismo nodo. Si tenemos 3 nodos (a, b y c) por visitar, entonces tendríamos una función de combinaciones sin repetición, es decir, tendríamos 6 posibles soluciones: abc, acb, bac, bca, cab, cba, para el caso de 4 nodos tendríamos 12 combinaciones, para 10 nodos tendríamos 90 combinaciones, para 100 ciudades tendríamos 9,900 combinaciones y así sucesivamente. Como ejemplo en el problema del Ulises de Homero que intenta visitar las ciudades descritas en la Odisea exactamente una vez (16 ciudades) donde existen múltiples conexiones entre las diferentes ciudades, Grötschel y Padberg (1993) llegó a la conclusión de que existen 653,837'184,000 rutas distintas para la solución de este problema. Simulated Annealing Simulated Annealing (SA) (recocido simulado, cristalización simulada, templado simulado o enfriamiento simulado) es un algoritmo de búsqueda metaheurística para problemas de optimización global; el objetivo general de este tipo de algoritmos es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande. A este valor óptimo se lo denomina "óptimo global". El nombre e inspiración viene del proceso de recocido del acero y cerámicas, una técnica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. El

calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales (un mínimo local de energía); el enfriamiento lento les da mayores probabilidades de recristalizar en configuraciones con menor energía que la inicial (mínimo global).

El método fue descrito independientemente por Scott Kirkpatrick, C. Daniel Gelatt y Mario P. Vecchi en 1983,2 y por Vlado Černý en 1985. El método es una adaptación del algoritmo Metropolis-Hastings, un método de Montecarlo utilizado para generar muestras de estados de un sistema termodinámico.

En cada iteración, el método de recocido simulado evalúa algunos vecinos del estado actual s y probabilísticamente decide entre efectuar una transición a un nuevo estado s' o quedarse en el estado s. En el ejemplo de recocido de metales descrito arriba, el estado s se podría definir en función de la posición de todos los átomos del material en el momento actual; el desplazamiento de un átomo se consideraría como un estado vecino del primero en este ejemplo. Típicamente la comparación entre estados vecinos se repite hasta que se encuentre un estado óptimo que minimice la energía del sistema o hasta que se cumpla cierto tiempo computacional u otras condiciones.

La probabilidad de hacer la transición al nuevo estado s es una función P(δ E, T) de la diferencia de energía δE=E(s')-E(s) entre los dos estados, y de la variable T, llamada temperatura por analogía con el concepto físico de temperatura.

Si δE es negativo, es decir, la transición disminuye la energía, el movimiento es aceptado con probabilidad P=1. Es importante remarcar que la condición de que el sistema siempre pase a un sistema de menor energía cuando se encuentra una no es en absoluto necesaria para el éxito del método. Cuando δE es positivo la probabilidad de transición P es siempre distinta de cero, aún , es decir, el sistema puede pasar a un estado de mayor energía (peor solución) que el estado actual. Esta propiedad impide que el sistema se quede atrapado en un óptimo local.

A medida que la temperatura tiende al mínimo, la probabilidad de transición a un estado de mayor energía tiende a cero asintóticamente. Cuando T llega a cero, el algoritmo solo aceptará cambios a estados con menor energía. Debido a esta propiedad, la temperatura juega un papel muy importante en el control de la evolución del sistema. A temperaturas altas, el sistema tenderá a saltos de energía grandes entre los estados, mientras que a temperaturas más bajas, los cambios en energía serán menores.

Así, en cada iteración el algoritmo tiende a encontrar estados con menor energía total. Hay muchas maneras de disminuir la temperatura, siendo la más usual la exponencial, donde T disminuye por un factor $\alpha<1$ en cada paso.

Los algoritmos heurísticos de este tipo realizan una búsqueda mas inteligente y eficiente, cabe mencionar que algo muy curioso de este algoritmo es que esta garantizado por una prueba matemática que con el suficiente tiempo y parámetros adecuados se puede llegar al optimo global, aunque por su puesto esto tomaría un tiempo considerablemente grande.

EQUIPO Y SOFTWARE UTILIZADO:

- Computadora personal (HP Omen con Core i7 y 8 Gb de RAM)
- El software de RapidMiner.

FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

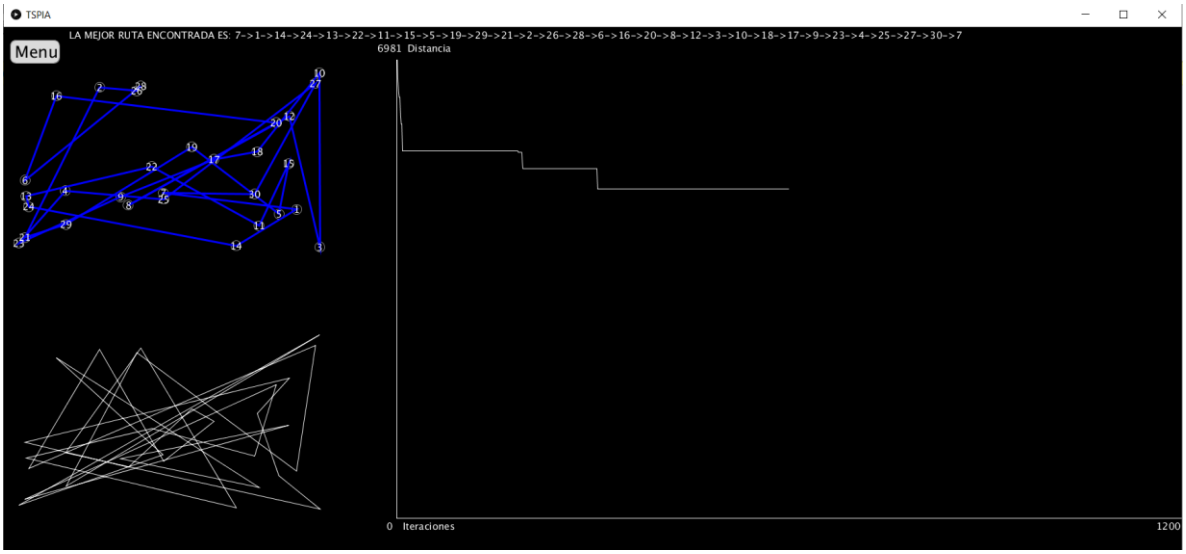
FUNCIONALIDAD DEL PROGRAMA:

A pesar de no ser parte original de la tarea me he tomado el atrevimiento de agregar la funcionalidad de generar una grafica que relaciona el como cambia la distancia con respecto al numero de iteraciones, esto con el propósito de analizar la evolución y cambio de la distancia, sobre todo el hecho que sobre sale en este algoritmo es el de que la distancia puede aumentar en bajo ciertas circunstancias determinadas por la estadística.

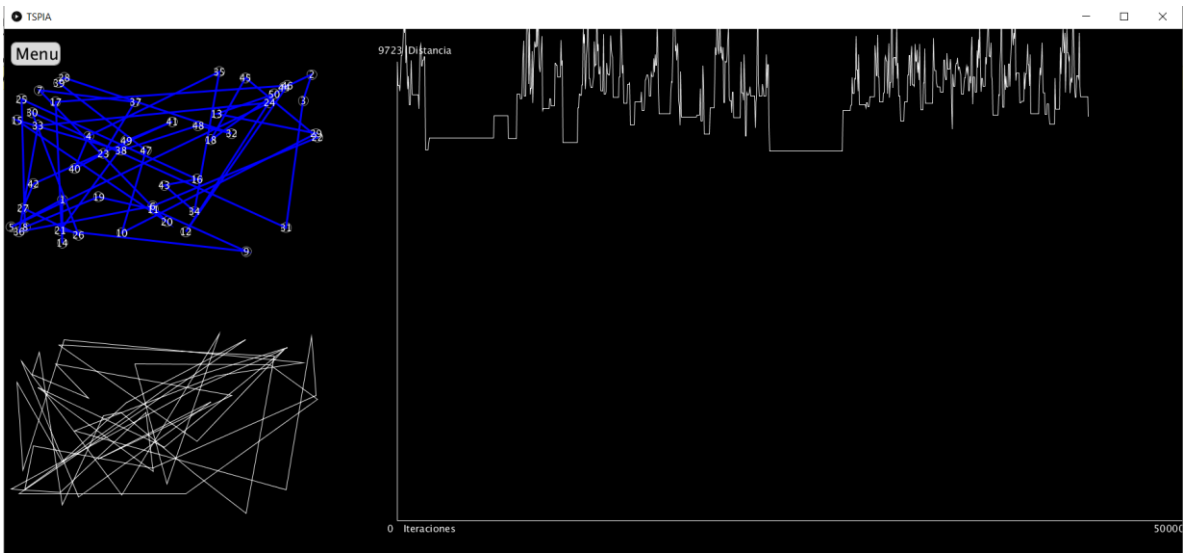
Para este primer ejemplo he usado un set de 30 ciudades generadas aleatoriamente, 2400 iteraciones, una temperatura inicial de 89 y un factor enfriamiento de 0.9999 esto con el fin de hacer totalmente obvio los posibles aumentos que pueden suceder en la distancia a lo largo de las iteraciones.

Para este ejemplo he usado parámetros mas conservadores, un set de 15 ciudades, 1200 iteraciones, una temperatura inicial de 30 y un factor de enfriamiento de 0.999, podemos ver una menor cantidad de repiques.

Para este ejemplo he implementado además un segundo criterio de parada para el algoritmo, en el caso de que después de cierto numero de iteraciones no haya una mejora el algoritmo se detendrá.



Atreviéndome un poco decidí usar parámetros relativamente grandes para experimentar el comportamiento y entender mejor el comportamiento y rol en el algoritmo de la temperatura inicial y el factor de enfriamiento, con un set de 50 ciudades y 50,000 iteraciones, temperatura inicial de 200 y un factor de enfriamiento de 0.99999, el programa tardo lo suyo en correr, pero es interesante el observar como va variando las decisiones que toma a cada iteración, realice pruebas con sets de ciudades similares, en la mayoría de los casos nunca se llega a siquiera acercar a los puntos de optimización a los que este algoritmo llega.

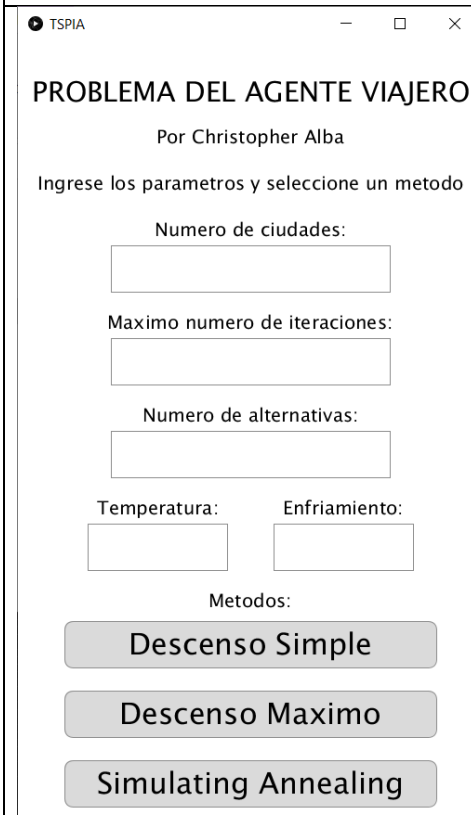


FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

INSTRUCCIONES DE USO:



Tome la decisión de construir este programa sobre las bases del anterior, por lo que su uso es realmente sencillo. Solo hace falta que se introduzcan los parámetros deseados y se seleccione el método, si uno de los parámetros no es correcto (letras en lugar de números, parámetros faltantes, etc.) se lanzara un mensaje de aviso y se limpiaran las entradas. Para usar las entradas basta con posicionar o dar click con el mouse sobre la casilla y teclear el numero deseado. Una vez que se oprima el botón del método que desee se ejecutará la visualización y en la esquina superior izquierda aparecerá un botón de menú el cual se puede usar para detener la simulación o salir una vez terminada.

Cabe mencionar que los cambios con respecto al anterior es la integración de un nuevo método y el generar el primer recorrido de forma aleatoria.

CODIGO RELEVANTE:

```
  void simulatingAnnealing() {
int a = int(random(numeroCiudades));
int b = int(random(numeroCiudades));
swap(recorrido, a, b);
distanciaRecorrida = distancia(ciudades, recorrido);
if (distanciaRecorrida<mejorDistancia) {
 mejorDistancia = distanciaRecorrida;
 arrayCopy(recorrido, mejorRecorrido);
} else {

 BigDecimal D = new BigDecimal(Math.exp((mejorDistancia-distanciaRecorrida)/temperatura));
 BigDecimal E = new BigDecimal(Math.random());

 if (E.compareTo(D) < 0) {
  mejorDistancia = distanciaRecorrida;
  arrayCopy(recorrido, mejorRecorrido);
```

```
  }
 }
 if (count<iteracionesMaximas) {
  temperatura *= enfriamiento;
 }
 count++;
 if (iteracionesMaximas == count) {
  textSize(15);
  fill(255);
  textAlign( LEFT, CENTER);
  String camino = "";
  for (int i = 0; i<numeroCiudades; i++) {
   camino += mejorRecorrido[i]+1;
   camino += "->";
   if (i==numeroCiudades-1) {
    camino += mejorRecorrido[0]+1;
   }
  }
  text("LA MEJOR RUTA ENCONTRADA ES: "+camino, 100, 10);
  noLoop();
 }
}
```

RETROALIMENTACIÓN:

Creo que lo mas notable a mencionar durante esta practica es sobre el uso de la “suerte” a nuestro favor, empezando con el hecho de que consideramos mejor el hecho de tener un recorrido inicial generado aleatoriamente a un recorrido en el orden numérico, esto debido a que en muy pocos casos el recorrido en orden numérico se encontrara en una buena posición en cuanto al coste del recorrido, mientras que un recorrido aleatorio al coste del riesgo de obtener un recorrido malo, podemos también obtener uno bueno. Además de que el método que hemos implementado en este programa utiliza un factor estadístico que es comparado con otro factor aleatorio para determinar el recorrido a tomar, esta decisión puede o llevarnos a casos mejores o a peores, sin embargo, debido a las propiedades estadísticas esto será beneficioso en la mayoría de los casos con los parámetros adecuados. Hablando de parámetros adecuados considero importante mencionar una de las complicaciones de implementar este algoritmo es que el numero aleatorio generado se encuentra entre uno y cero, al igual que la expresión probabilística, y es bien sabido el problema que se genera con los numero de punto flotante en las ciencias de la computación debido a la impresión de la representación binaria de los números decimales, por lo que después de cierta cantidad de iteraciones, al estos números volverse pequeños del orden 1x10^-15 la representación binaria es realmente errónea y las comparaciones no se realizan efectivamente.
En cuanto a los parámetros de temperatura y el factor de enfriamiento, considero que en el caso de ser ambos altos (en sus respectivos rangos) tendremos una alta inestabilidad que nos llevara a resultados no óptimos del todo, en el caso de ambos ser bajos, tendremos algo muy parecido a los métodos de descenso, mientras que las otras dos variantes resultan las más interesantes.

BIBLIOGRAFÍA:

- Apuntes del profesor y videos de clase.
- Applegate, D., Bixby,R., Chvatal V., Cook,W. “On the solution of the Traveling Salesman
- Problem”. DocumentaMathematica-Extra Volume ICM III. 1998. 645-656.
- https://es.wikipedia.org/wiki/Algoritmo_de_recocido_simulado
- Gutiérrez Andrade, Miguel Ámgel; de los Cobos Silva, Sergio Gerardo; Pérez Salvador, Blanca Rosa (junio de 1998). «Optimización con recocido simulado para el problema de conjunto independiente». En Línea² (Universidad Autónoma Metropolitana) 3. Archivado desde el original el 4 de diciembre de 2011. Consultado el 29 de julio de 2011.

Las bases del código e inspiración de este fueron extraídas del siguiente GitHub:

- https://github.com/CodingTrain/website/blob/main/CodingChallenges/CC_035.1_TSP/P5/sketch.js

siendo esto un trabajo en JavaScrip realizado por “The Coding Train” (https://thecodingtrain.com) con las librerías de P5.js