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: *BÚSQUEDA HEURÍSTICA*

OBJETIVO:
Implementar la solución al problema del agente viajero por los métodos de descenso simple y descenso máximo limitado, usando una interfaz gráfica para visualizar el proceso de solución además del estado inicial y final de la ruta y puntos de visita.

RESUMEN:
Dado los parámetros iniciales del numero de ciudades, número máximo de iteraciones y numero de propuestas (si es el caso del método de máximo descenso) se calculará la ruta más optima encontrada según los métodos de descenso simple y máximo descenso limitado, 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. TSP se encuentra clasificado como Problema de optimización Combinatoria, es decir, es un problema donde intervienen cierto número de variables donde cada variable puede tener N diferentes valores y cuyo número de combinaciones es de carácter exponencial, lo que da lugar a múltiples soluciones óptimas (soluciones que se calculan en un tiempo finito) para una instancia. TSP es un problema considerado difícil de resolver, denominándose en lenguaje computacional NP-Completo, es decir, es un problema para el que no podemos garantizar que se encontrará la mejor solución en un tiempo de cómputo razonable. Para dar solución se emplean diferentes métodos, entre los cuales, los principales se denominan heurísticas cuyo objetivo es generar

soluciones de buena calidad en tiempos de cómputo mucho más pequeños (soluciones óptimas tiempo – respuesta).

Algoritmos heurísticos

Los principales algoritmos heurísticos que se han utilizado en programación y secuenciamiento de operaciones se engloban en esta sección y están basados en métodos de búsqueda llamados de "vecindario". Inicialmente la técnica de Búsqueda Local se comenzó a utilizar cuando se buscaba mejorar los recorridos (tours) para el problema del agente viajero sustituyendo aristas (ciudades) de la trayectoria por otras. En programación y secuenciamiento de operaciones Nicholson (1967) fue uno de los primeros en proponer el uso de la búsqueda local. El intercambiada un trabajo j definido en la secuencia y lo insertaba en alguna otra posición para minimizar el makespan en una configuración flowshop de tres máquinas. Sin embargo, los métodos de búsqueda local no fueron muy atractivos hasta mucho más tarde. Las razones posiblemente eran: las computadoras no eran lo suficientemente rápidas para una adecuada búsqueda en problemas prácticos, y los métodos eran muy simplistas como para considerarlos como una contribución importante.

El recocido simulado como técnica de optimización fue el instrumento para establecer a la búsqueda local como área de investigación prospera.

El más simple algoritmo de búsqueda es denominado en descenso donde repetidos intentos son hechos para mejorar la solución actual. Los intentos en mejorar involucran perturbaciones a la actual solución, si la perturbación nos lleva a una mejor solución un movimiento es hecho hacia la nueva solución. Un vecino define que posibles movimientos son permitidos. La búsqueda continúa hasta que ya no es posible mejorar sobre la actual solución. En una búsqueda con inicio multidescenso se ejecuta el algoritmo en descenso varias veces. Usando un diferente punto de solución puede ayudar a producir mejores soluciones.

Existen estrategias bien conocidas que permiten progresar para evadir soluciones pobres y atravesar en el espacio para encontrar mejores soluciones. En Recocido Simulado, las soluciones pobres son aceptadas con una probabilidad de aceptación de $e^{-\Delta/T}$, donde Δ es la cantidad en la cual la función objetivo empeora con la nueva solución y T es un parámetro llamado temperatura que es reducido durante la ejecución del algoritmo acorde a un programa de enfriamiento. En Búsqueda Tabú, un movimiento en el vecindario es hecho aun si la solución es pobre y para lograr direccionar la búsqueda en diferentes zonas del espacio de solución, una lista tabú almacena propiedades de los puntos de la solución recién visitados. Los posibles movimientos que conducen a soluciones con propiedades ya almacenadas son prohibidos.

Los métodos que utilizaremos en este programa son los de descenso simple y de descenso máximo limitado, estos básicamente consisten en la misma base la cual es escogiendo dos nodos hacer que cambien la ruta entre ellos y si la ruta resultante tiene un coste menor que el original entonces se toma, de lo contrario se descarta y se prueba otro intercambio, la única diferencia entre ellos es que el descenso máximo limitado además tiene candidatos por cada intercambio a partir de un pivote que intercambia con distintos nodos, evalúa cual es el mejor y realiza lo mismo que antes.

EQUIPO Y SOFTWARE UTILIZADO:

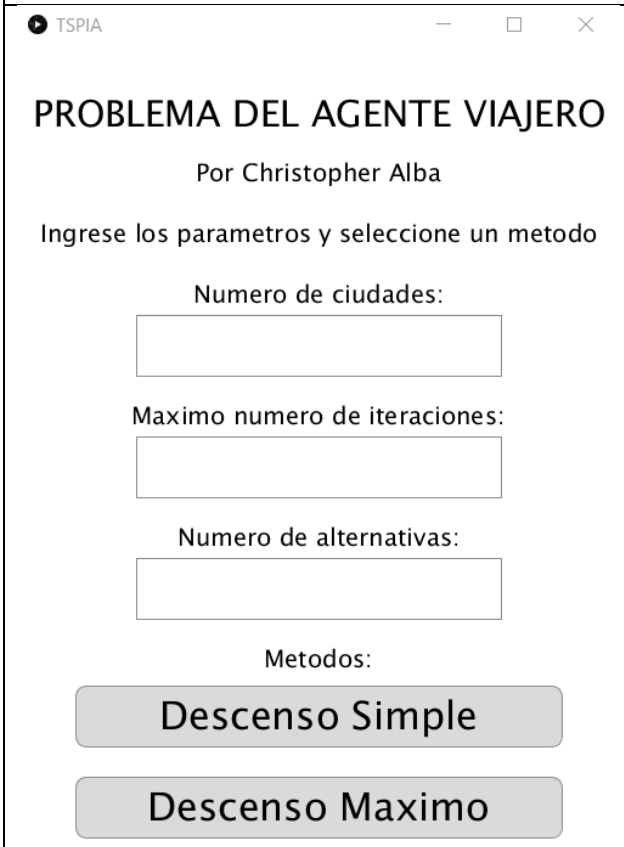
- Computadora personal (HP Omen con Core i7 y 8 Gb de RAM)
- El software de RapidMiner.

FACULTAD DE INGENIERÍA

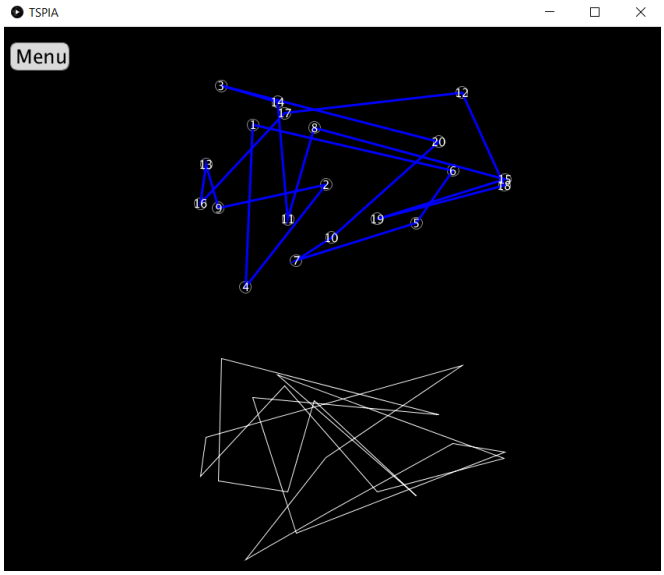
FORMATO
PRÁCTICAS DE LABORATORIO

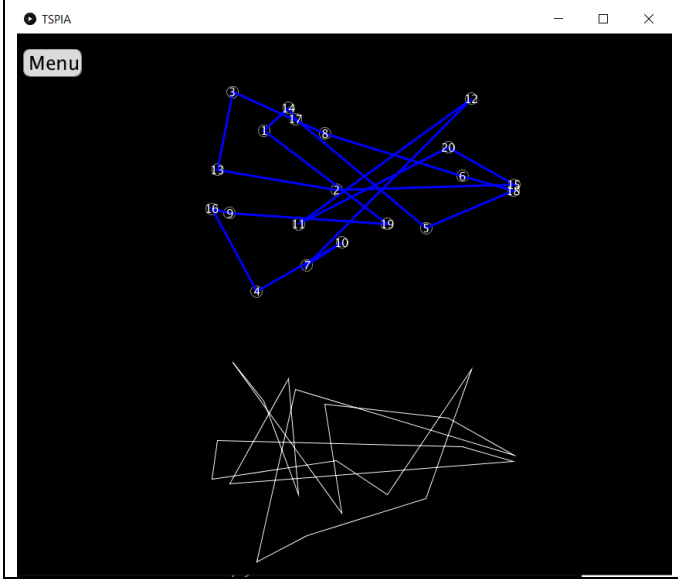
UNIVERSIDAD AUTÓNOMA DE CAMPECHE

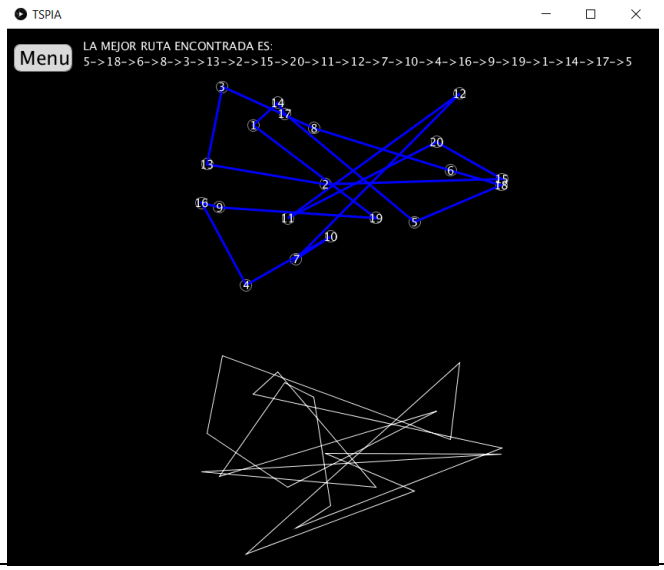
FUNCIONALIDAD DEL PROGRAMA:



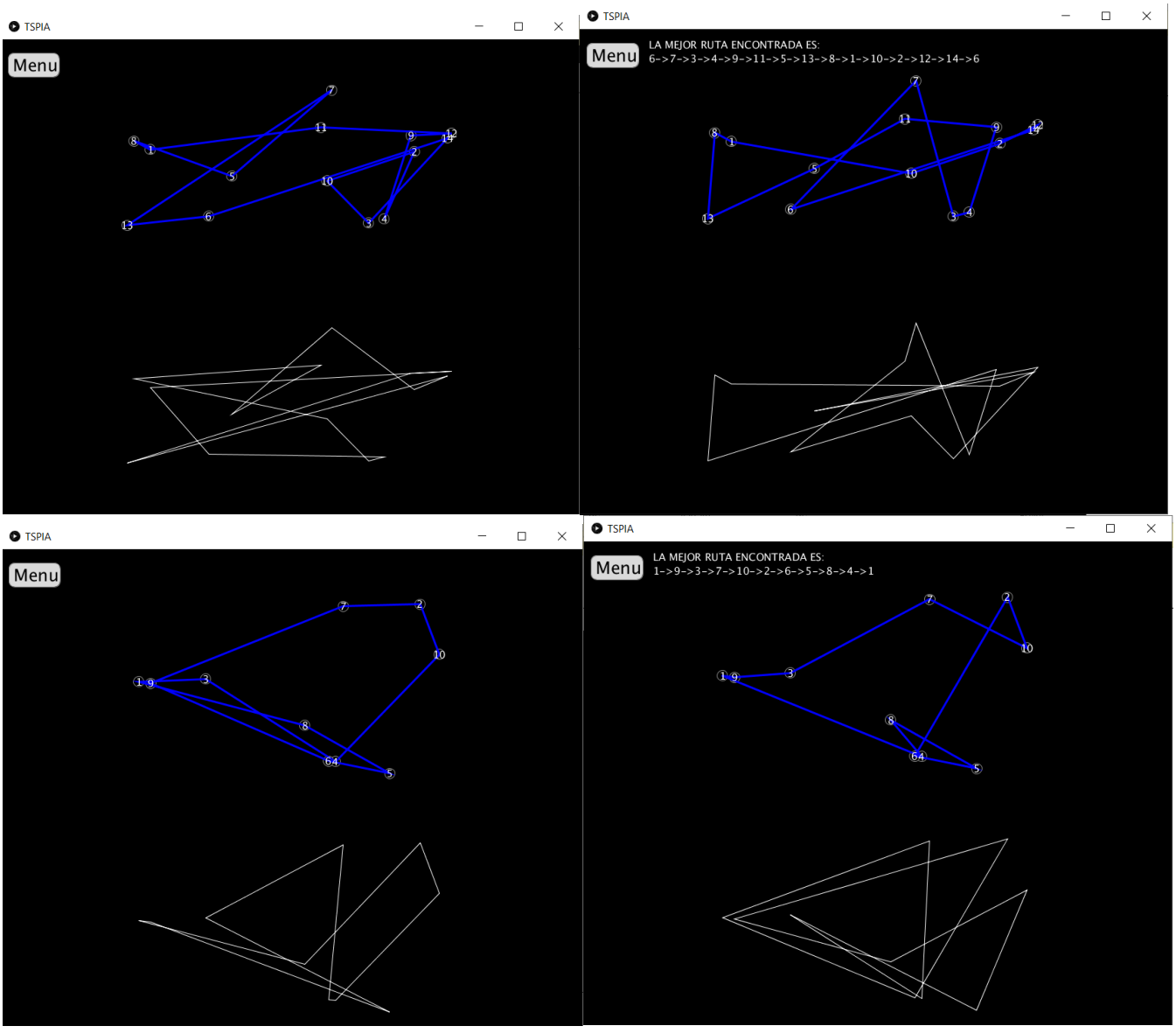
Una vez que introducimos los parámetros el programa nos dirigirá a la visualización del problema, en azul podemos observar el mejor camino encontrado hasta el momento, tanto los caminos como los vértices, en blanco los caminos evaluados, podemos observar como cambia hasta el momento que o llega al número de iteraciones máximas o no encuentra mejor ruta, y se muestra la mejor ruta

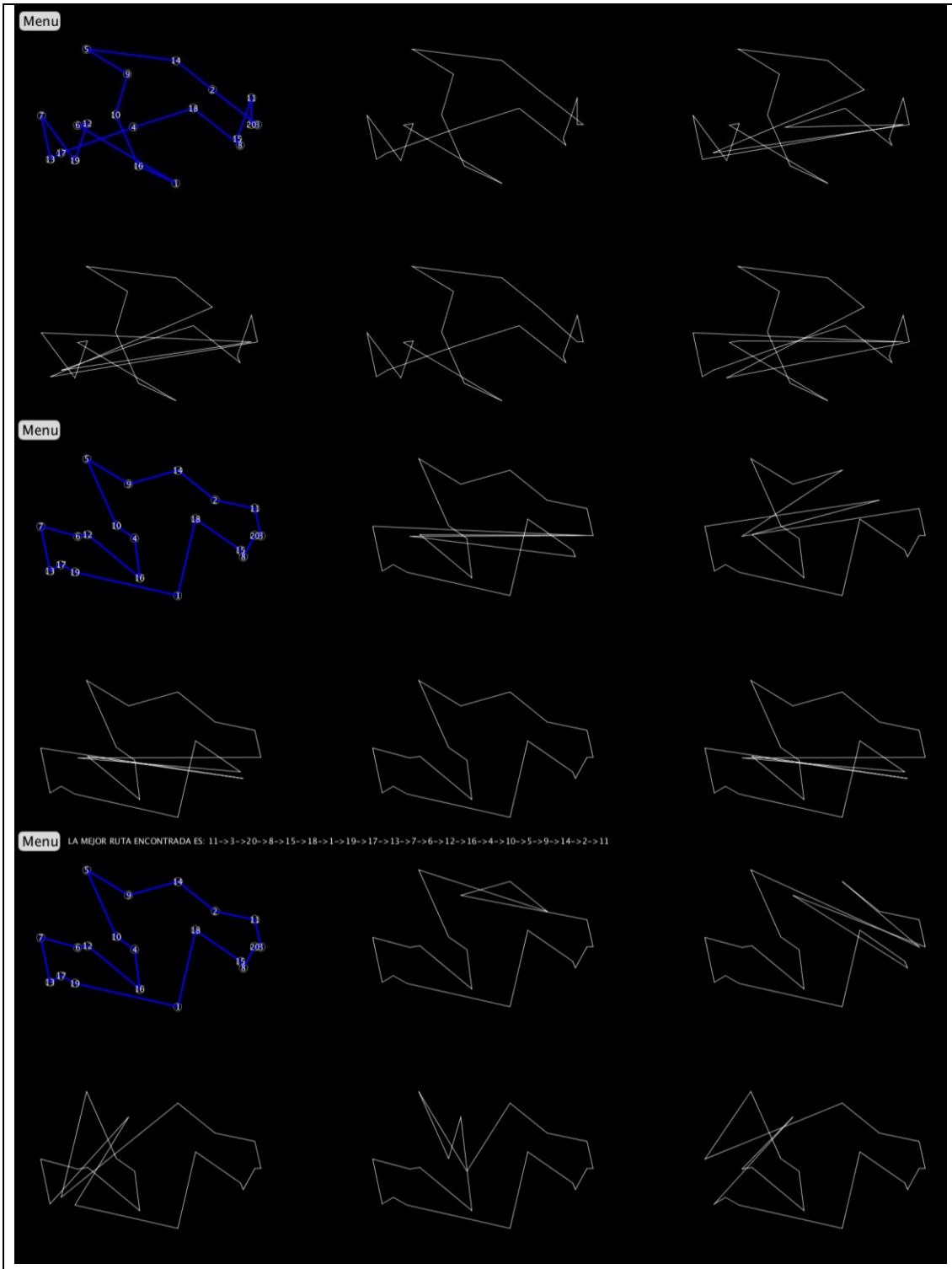




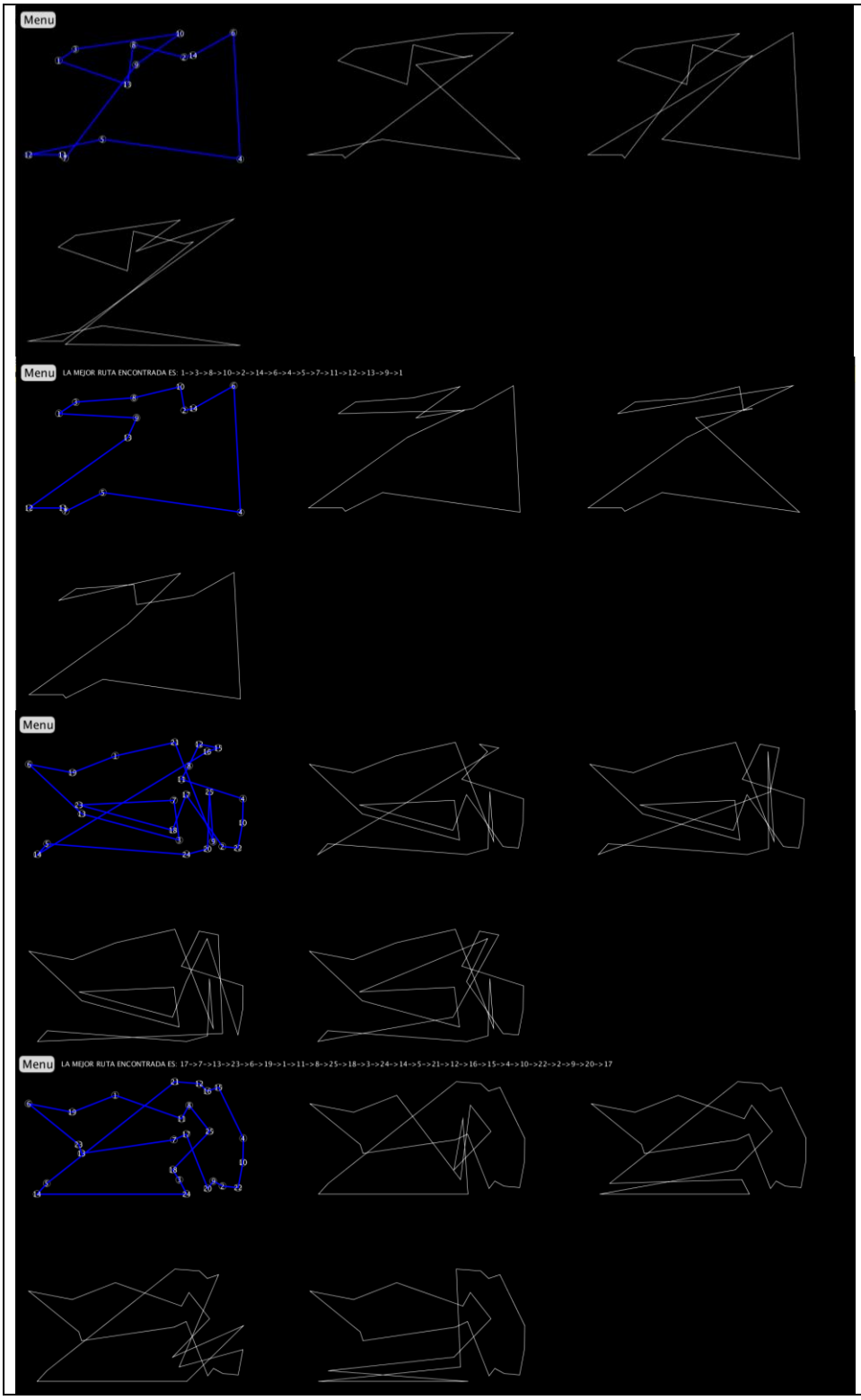


Aquí podemos ver dos ejemplos de ejecución variando los parámetros tanto en la cantidad de ciudades como en el número de iteraciones realizadas. Debemos de recordar que la posición en donde se encuentran las ciudades y por lo tanto la distancia entre ellas es determinada aleatoriamente como puntos en una cuadricula, por lo que a veces se darán casos extremadamente simples, además de la velocidad de las iteraciones puede resultar algo difícil ver los cambios a detalle



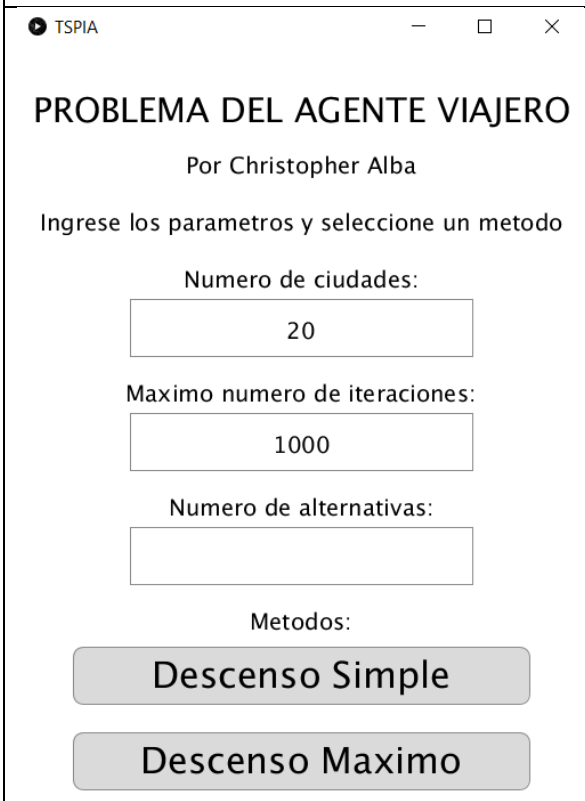


Aquí tenemos la interface a la que nos lleva en el caso de que seleccionemos el método de máximo descenso, cabe mencionar que podemos decir cuantas propuestas deseamos y solo esas serán mostradas y evaluadas.



Ejemplos variando los parámetros tanto la cantidad de ciudades como el número de propuestas en cada una. Inclusive atreviéndome a poner 25 ciudades, en realidad realice pruebas y el programa soporta números realmente grandes y encuentra rutas bastante coherentes con los parámetros adecuados, siendo que legue a encontrar la tercera ruta mas optima para un conjunto predefinido de 49 ciudades, sin embargo, puede acusárseme de no ser parcial puesto que debemos de recordar que una gran parte de la efectividad de estos métodos depende de la ruta inicial, la cual yo ayude a crear a “ojo humano”.

INSTRUCCIONES DE USO:



El programa es realmente sencillo de usar, 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.

CODIGO RELEVANTE:

```
//función para calcular la distancia euclidiana entre dos ciudades
 float distancia(PVector[] ciudades, int[] recorrido) {
  float total = 0;
  PVector ciudadA, ciudadB;
  for (int i = 0; i < recorrido.length - 1; i++) {
    ciudadA = ciudades[recorrido[i]];
    ciudadB = ciudades[recorrido[i + 1]];
    total += dist(ciudadA.x, ciudadA.y, ciudadB.x, ciudadB.y);
  }
  return total;
}
```

```
//Función que realizara los intercambios entre las ciudades para el método de descenso simple
void swap(int[] recorrido, int pivote, int cambio) {
  int temp = recorrido[pivote];
  recorrido[pivote] = recorrido[cambio];
  recorrido[cambio] = temp;
}
//Función que realizara los intercambios entre las ciudades para el método de máximo descenso
void swapPropuesta(int propuesta[], int a) {
  int b = int(random(numeroCiudades));
  arrayCopy(mejorRecorrido, propuesta);
  swap(propuesta, a, b);
  distanciaRecorrida = distancia(ciudades, propuesta);
  if (distanciaRecorrida<mejorDistanciaPropuesta) {
    mejorDistanciaPropuesta = distanciaRecorrida;
    arrayCopy(propuesta, mejorRecorridoPropuesto);
  }
}
//Algoritmo del método de descenso simple
void descensoSimple() {
  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);
  }
  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:", 100, 20);
    text(camino, 100, 40);
    noLoop();
```

```
  }
}

//Algoritmo del método de máximo descenso limitado, en este caso a 5
void maximoDescenso() {
  int a = int(random(numeroCiudades));

  if (numeroPropuestas>=1) {
    swapPropuesta(propuesta1, a);
  }
  if (numeroPropuestas>=2) {
    swapPropuesta(propuesta2, a);
  }
  if (numeroPropuestas>=3) {
    swapPropuesta(propuesta3, a);
  }
  if (numeroPropuestas>=4) {
    swapPropuesta(propuesta4, a);
  }
  if (numeroPropuestas>=5) {
    swapPropuesta(propuesta5, a);
  }

  arrayCopy(mejorRecorridoPropuesto, mejorRecorrido);
  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, 35);
    noLoop();
  }
}
```

FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

RETROALIMENTACIÓN:

El problema del Agente Viajero (TSP) es un problema cuya solución ha sido estudiada desde los inicios de la Inteligencia Artificial considerando que su aplicación puede ser en cualquier área de estudio cuyos problemas reflejen una situación donde se tienen diferentes puntos a visitar con un costo considerado en el enlace entre dichos puntos (costo: recursos empleados como distancia, tiempo, monto económico, etc.). Cada autor ha propuesto soluciones para ciertas instancias de TSP, cada uno con una perspectiva diferente empleando técnicas que no son repetibles pero que, en determinado momento, se pueden emplear para dar lugar a nuevas soluciones; de las técnicas empleadas la más común es el uso de redes neuronales dada su similitud, donde cada neurona es un nodo a visitar y las relaciones entre neuronas es el vector que representa el costo solo por mencionar un ejemplo de estos.
El tener una idea visual es sin duda de gran ayuda en el camino de entender la verdadera intuición y profundo entendimiento de los conceptos y retos que nos trae el resolver esta clase problemas.

RECOMENDACIONES ADICIONALES:

El código que escribí puede ser optimizado en algunos puntos además de ser adaptado para una mayor flexibilidad, un ejemplo de esto es al momento de implementar el método de máximo descenso se utilizan estructura de tipo if para determinar y ejecutar el numero de propuestas requerido, al igual que solo esta hecho par aun máximo de 5, y carece de abstracción en esa parte, una mejora que surge a la vista al momento es el uso de un array de arrays de recorridos propuestos con lo que se matan dos pájaros de un tiro, tanto el hecho de no poder tener mas de 5 así como eliminar las estructuras if sustituyéndola por un for y haciendo todo mas completo y autocontenido, sin embargo las librerías que use para la ejecución tienen algunos problemas con las matrices ya que sobrescriben algunos métodos de estas y crean otros, cosa que puede ser beneficiosa y perjudicial a la vez, de igual forma usando clases y herencia así como sobrecarga se puede reducir tanto el numero necesario de funciones como la cantidad de código escrito.
Sin embargo, debido a ser un ejercicio puntual se prefirió el uso de una sola clase cosa que limita estas características pero hace más sencillo la acción de codificar el problema.

BIBLIOGRAFÍA:

- Apuntes del profesor y videos de clase
- Rodríguez, R. P. (2014). Programación de operaciones en configuraciones jobshop flexibles utilizando un algoritmo de estimaciones de distribuciones.
- Applegate, D., Bixby,R., Chvatal V., Cook,W. “On the solution of the Traveling Salesman Problem”. DocumentaMathematica-Extra Volume ICM III. 1998. 645-656.

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)