Tarea Programada

Vendedor Viajero

Descripción:

El problema del vendedor viajero (Travelling Salesman) trata sobre un vendedor que quiere visitar una lista de ciudades específicas. Para su recorrido quiere recorrer la menor distancia total posible y además desea visitar cada ciudad una única vez con excepción de la primera a la cual quiere regresar al final de su travesía.

Este problema ha sido muy estudiado en la computación debido a su relevancia en el mundo real: desde la selección de trayectos para un Uber, a decisiones con respecto al uso de vehículos de distribución de correo y problemas logísticos de aerolíneas.

Además de relevante, el problema es de reconocida dificultad. Esto porque para realmente estar seguros de contar con una ruta óptima sería necesario probar todas las combinaciones posibles de trayectos lo cual le da al problema una complejidad computacional O(n!) (un tiempo muy superior a polinomial). Con solo 15 ciudades habría que revisar 1 307 674 368 000 posibles trayectos.

Para enfrentarlo los informáticos han planteado variedad de algoritmos: desde árboles de búsqueda hasta la utilización de heurísticas. Sin embargo, una de las soluciones más interesantes está fuertemente vinculada al área de inteligencia artificial: el algoritmo genético.

Un algoritmo genético es un tipo de algoritmo que se basa en la simulación del proceso de selección natural que ocurre en la naturaleza para llegar a una solución (relativamente) óptima a un problema. Y puede ser descrito de la siguiente manera:

1. Se inicia con una población inicial de *n* individuos aleatorios: cada individuo codifica una posible solución al problema (en este caso, un posible orden de visitar las ciudades) generada al azar. El tamaño de la población es un parámetro del algoritmo que aumenta el costo computacional pero permite una mayor exploración de soluciones.
2. Para cada individuo de la población se aplica un algoritmo de evaluación (*fitness*) que obtiene un puntaje numérico de la "calidad" de su solución (en este contexto dicha calidad estaría directamente relacionada a las distancias recorridas por la solución del individuo).
3. Los individuos se ordenan según su evaluación de aptitud, poniendo a los más "aptos" de primero.
4. Se inicia la creación de una nueva generación poblacional:
    a. Los individuos más aptos de la generación anterior sobreviven y pasan a la siguiente generación (*elitism/hall of fame*). El porcentaje de sobrevivientes es un parámetro del algoritmo, pero se recomienda un porcentaje bajo para facilitar el proceso evolutivo.
    b. Se crea un porcentaje de individuos nuevos completamente aleatorios (*spontaneous generation*) por generación espontánea. El porcentaje de individuos espontáneos por generación es un parámetro del algoritmo, pero

se recomienda un porcentaje bajo. La generación de individuos aleatorios permite introducir variabilidad a las soluciones del problema.

c. Para el resto (1 - *elitism* - *spontaneous*) de los individuos faltantes de la nueva generación ocurre lo siguiente:
   i. Se eligen al azar dos padres de la generación anterior (*selection*), esta selección aleatoria idealmente está sesgada, eligiendo con mayor probabilidad los individuos más aptos.
   ii. Se utiliza un algoritmo de mezcla (*crossover*) que mezcla las soluciones de ambos padres. Esta mezcla idealmente posee información proveniente de ambos padres.
   iii. Luego el individuo creado tiene una probabilidad *m* de mutar (*mutation*), este valor *m* es un parámetro del algoritmo. El propósito de las mutaciones es que incluso si el "pozo genético" se empieza a volver uniforme, haya posibilidad de mejorar utilizando las soluciones ya existentes. (Para decidir si un individuo debe mutar genere un número al azar en un rango específico y compárelo con *m*).
d. Una vez que se ha creado la nueva generación, esta sustituye a la anterior. Este proceso generacional se conoce dentro del algoritmo como una época (*epoch)*.
e. El algoritmo se repite volviendo a ejecutar del paso 2 en adelante.
f. Conforme pasan las épocas, los individuos de la población van presentando cada vez mejores soluciones.

Current progress

Tarea programada:

En esta tarea el objetivo consiste en implementar de manera adecuada un algoritmo genético para solucionar el problema del vendedor viajero. Para ello deberá utilizar lo aprendido sobre herencia y polimorfismo para implementar los métodos faltantes de la clase GeneticAlgorithm proveída por su profesor.

La clase GeneticAlgorithm se encuentra en un archivo header (.h/.hpp) que se utiliza generalmente para declarar clases aparte de su implementación. Para implementarla deberá crear un archivo GeneticAlgorithm.cpp que importe el encabezado y definir las funciones:

```
tipoRetorno GeneticAlgorithm::nombreFuncion(params){ ...definición...
}
```

Deberá implementar las siguientes funciones de la clase GeneticAlgorithm:

- `Constructor(params...)`: Implemente un constructor generalizado para la clase GeneticAlgorithm. Considere que cualquier algoritmo genético heredará este constructor por lo tanto debe ser general.
- `Destructor(params...)`: Implemente el destructor generalizado para la clase GeneticAlgorithm.
- `unsigned int getEpoch()`: Retorna el número de la época actual.
- `virtual void reset(params...)`: Reinicia el algoritmo genético, regresando a la primera época y modificando los parámetros del algoritmo.

Una vez implementados los métodos de la clase GeneticAlgorithm, podrá crear una clase heredera de la misma: la clase TravellingSalesman. Para ello deberá implementar los métodos virtuales de la clase, así como incluir aquellos atributos que sean necesario para la ejecución del algoritmo.

- `void reset(param...)`: Llama al método reset de la clase padre y reinicia la población del algoritmo genético. Recuerde no tener fugas de memoria.
- `void epoch()`: Ejecuta una época del algoritmo descrito previamente creando una nueva generación. Recuerde no tener fugas de memoria.
- `float getBestIndividual()`: Obtiene y retorna la distancia total de recorrido del mejor individuo de la población.
- `void drawBestIndividual(float** points,unsigned int &numPoints)`: Obtiene la codificación del mejor individuo y lo convierte a una representación de puntos en un plano cartesiano (x,y). En este caso, los puntos coinciden a las coordenadas de las ciudades a visitar en el orden en el que se visitarán. La variable numPoints se modifica a la cantidad de ciudades que forman parte del recorrido.

A parte de ello puede considerar implementar otros métodos que ayuden a organizar y dividir el problema, como por ejemplo: El método *selection* para seleccionar a los padres de las nuevas generaciones, un método *crossover* que crea un nuevo individuo mezclando la información de sus padres, el método *mutation* que muta aleatoriamente a un individuo y la función *fitness* para evaluar a los individuos de la población. Puede considerar otros métodos que le sean de utilidad.

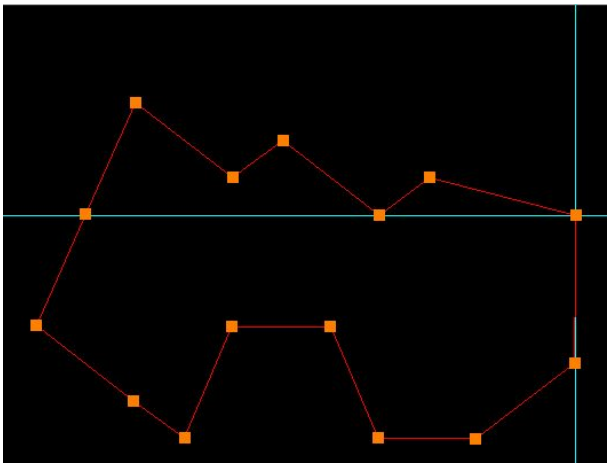
Usted tiene libertad para decidir el algoritmo de evaluación (*fitness*), selección (*selection*), mezcla (*crossover*) y mutación (*mutation*). Puede probar diferentes estrategias y descubrir cuál tiene mejores rendimientos/cuál converge más rápido a una solución. Es válido investigar posibles soluciones en Internet, sin embargo, recuerde que no se permite copiar código, por lo que aunque realice investigación deberá implementar el código usted mismo.

Con la clase implementada TravellingSalesman le será posible crear un código main para probar el algoritmo.

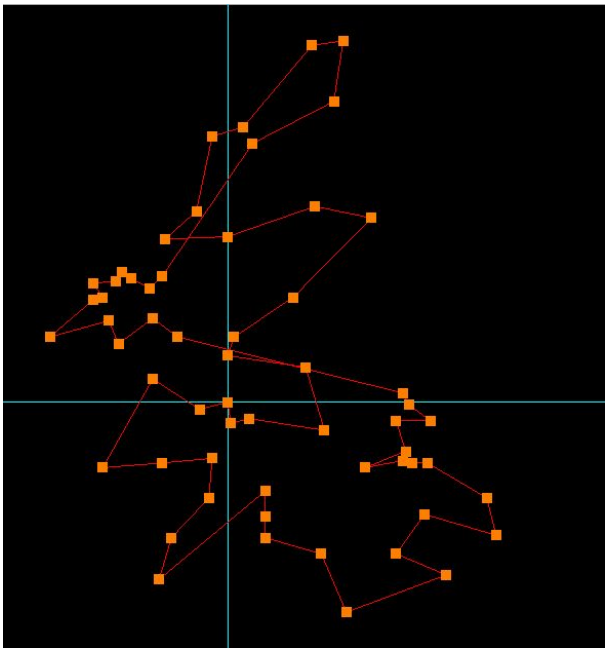
Los casos de prueba se encuentran disponibles en este documento, con incrementos de dificultad significativos entre cada uno de los tres casos. Los arreglos presentados corresponden a las coordenadas (x,y) de las diferentes ciudades con lo cual le será posible calcular la distancia euclidiana entre cada una de las ciudades.

Para esta tarea programada puede trabajar solo o en parejas, en cuyo caso debe incluir el nombre de ambos participantes.

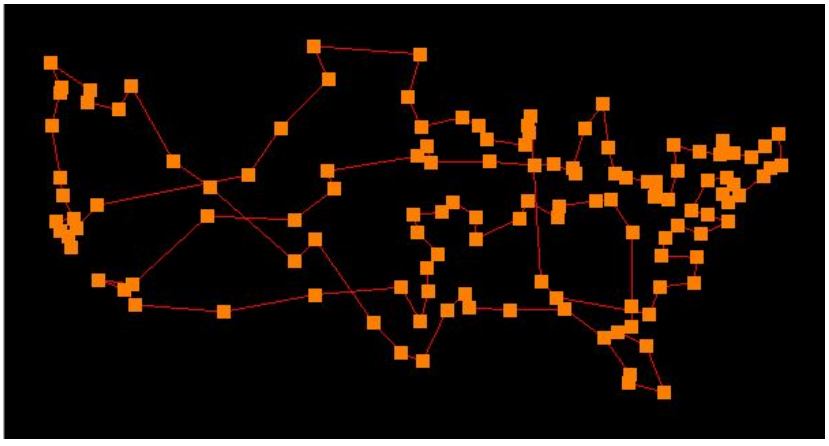
Las soluciones para cada uno de los casos son:
LAU15: 284.38 (solución óptima)



WG59: 1156.72 (solución subóptima)



SGB128: 21312.85 (solución subóptima)



Si su algoritmo encuentra una solución superior a las presentadas, entonces habrá ganado exitosamente muchos puntos de honor. Recuerde que la nitidez y optimización del código también benefician a que la ejecución sea más rápida, lo que permite ejecutar más épocas en menor tiempo.

(Puntos de logro/puntos opcionales +10): Refine su algoritmo de manera que logre lo siguiente: utilizando una población de máximo 200 individuos y un máximo de 1 millón de épocas (1000000 epochs) encuentre la solución óptima para el caso LAU15, una solución con distancia menor a 1400 para el caso WG59 y una solución con distancia menor a 25000 para SGB128. No se vale hacer trampa (incluir soluciones precalculadas en su población), sino que su población debe converger naturalmente a la solución. Para estabilizar lo aleatorio del experimento recuerde que puede "sembrar" la aleatoriedad por medio del comando `srand(n)`, para que la aleatoriedad produzca siempre el mismo resultado y hacer que su experimento sea repetible.

Si implementó el algoritmo de acuerdo a las especificaciones, podrá utilizar el código proveído por su profesor en el siguiente link para ver sus resultados obtenidos. Sin embargo, deberá modificar las líneas 11 y 89 acorde a su archivo/nombre de clase.

De estar utilizando wxDev-Cpp debería poder compilar el código sin problemas al incluirlo a un proyecto. Si no cuenta con wxDev-Cpp entonces necesitará instalar la biblioteca wxWidgets y OpenGL. Dependiendo de su IDE, este ya contará con las bibliotecas o tendrá un manejador de paquetes para instalarlo. Si no está utilizando un IDE con dichas capacidades, hay muchas guías de como instalarlos en Linux. Por otro lado, en el sistema Windows la instalación es más complicada, pero le recomiendo buscar MSYS2, descendiente del proyecto Minimal GNU for Windows (MINGW) que le permitirá instalar un compilador gcc/g++ de 64 bits en Windows. Además incluye una terminal que le permitirá instalar bibliotecas con "facilidad" (nunca es realmente fácil) por medio del comando pacman. En todo caso, esta etapa es completamente opcional, entonces no se complique de no ser necesario.

Para compilar el código mediante terminal utilice el comando:

```
g++ GeneticAlgorithmVisualizer.cpp -o GeneticAlgorithmVisualizer -O2
-I"include_path" -L"lib_path" -l:libwxmsw31u.a -l:libwxmsw31u_gl.a
-lopengl32
```

-I"include_path" le permite incluir el camino hacia la carpeta include de su compilador y -L"lib_path" le permite incluir el camino hacia la carpeta lib de su compilador, asegúrese de que en dichas carpetas se encuentren las bibliotecas necesarias para la compilación.
Modifique los nombres de las bibliotecas wx y openGL según la versión instalada en su computadora.

Guía no oficial:

Si desea utilizar MSYS2 para manejar sus bibliotecas y compilador deberá instalarlo de la página del programa. Podrá entonces abrir la terminal MSYS2 como cualquier programa.

Luego debe ejecutar el comando:

```
pacman -S mingw-w64-x86_64-gcc
```

Para instalar los binarios necesarios para compilar programas en 64 bits (en la carpeta mingw64/bin debería encontrar gcc, g++ y muchos otros). Sería recomendable que agregue esta carpeta en sus variables de ambiente para poder llamar gcc/g++ y los demás programas desde la terminal.

Luego podrá instalar la biblioteca wx: www.wxwidgets.org/downloads/

Sin embargo, es necesario compilar la biblioteca para que funcione por lo que dentro del CMD deberá dirigirse a la carpeta de instalación de wx llamada /build/msw (MicroSoft Windows) y ejecutar el comando:

```
mingw32-make -j4 -f makefile.gcc SHARED=1 UNICODE=1 BUILD=release MONOLITHIC=1
LDFLAGS=-Wl,--allow-multiple-definition
```

De no poder ejecutar es porque no tiene instalado el programa mingw32-make o no lo tiene dentro de sus variables de ambiente en cuyo caso deberá poner el camino directo al programa. Esto deberá compilar las bibliotecas. Copie el directorio include/wx a su directorio include dentro de mingw64. Luego copie las bibliotecas y dll's generados de la carpeta lib. Sus programas requerirán al menos uno de los .dll en el mismo directorio que el ejecutable para ejecutar.

La biblioteca de openGL viene incluida con el paquete glut, el cual puede instalar utilizando el comando:

```
pacman -S mingw-w64-x86_64-freeglut
```

desde la terminal de MSYS2. Con esto debería contar con todas las bibliotecas y herramientas para poder compilar y ejecutar programas que utilicen OpenGL y wxWidgets en sistemas Windows, sin depender de ningún ambiente de desarrollo.