Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Laboratorio de Algoritmos y Estructuras II - CI2692

Un algoritmo divide-and-conquer para el problema del agente viajero

Proyecto 1

Profesor: Guillermo Palma
Estudiantes:
Haydeé Castillo Borgo. Carnet: 16-10209
Jesús Prieto. Carnet: 19-10211

Trimestre Abril-Julio 2023

1. Introducción

El problema del agente viajero, TSP por sus siglas en inglés, es un problema de optimización ampliamente estudiado en matemáticas y en computación. En este problema se tiene un conjunto de ciudades y se busca diseñar un tour que permita recorrer todas las ciudades sin repetición y en la menor distancia posible.

En el presente proyecto se considera el problema del TSP Euleriano, que es aquel en el cual las ciudades vienen dadas como puntos del plano cartesiano; y se implementa un algoritmo divide-and-conquer en conjunto con un algoritmo de optimización 2-opt, para resolver dicho problema y presentar su solución de acuerdo al esquema de TSPLIB.

A continuación se describen los algoritmos implementados, los resultados obtenidos y se presentan las conclusiones alcanzadas en la realización del proyecto.

2. Diseño de la solución

Para implementar el algoritmo de divide-and-conquer se emplea el lenguaje de programación Kotlin y se organiza el código de acuerdo a 4 algoritmos principales:

- Algoritmo 1: divideAndConquerTSP, el cual representa el centro del esquema divide-and-conquer empleado para resolver el problema TSP
- Algoritmo 2: obtenerParticiones, el cual se encarga de dividir el problema en instancias más pequeñas para poder resolverlo
- Algoritmo 3: combinarCiclos, el cual se encarga de unir las soluciones dadas para las instancias pequeñas en una solución completa para una instancia más grande
- Algoritmo 4: divideAndConquerAndLocalSearchTSP, el cual se encarga de modificar la solución proporcionada por el algoritmo divideAndConquerTSP a través del algoritmo de optimización 2-opt y propocionar una solución más eficiente al problema del TSP.

Además se tienen el Algoritmo 0, denominado obtenerDatosTSP, el cual se encarga de registrar y extraer los datos del problema TSP proporcionados a través de un archivo con formato TSPLIB; y el Algoritmo 5, denominado generarArchivoSolucionTSPLIB, el cual se encarga de procesar la solución proporcionada por los algoritmos 1-4 y generar un archivo en formato TSPLIB.

En esta implementación, el conjunto de ciudades se denota P y viene dado por un arreglo de tripletas que contienen la coordenada x (como un Double), la coordenada y (como un Double) y el número respectivo de la ciudad (como un Int). Por otro lado, los tours vienen representados como ciclos que contienen los lados de las ciudades y se organizan en arreglos de pares de dichas tripletas, pues cada lado se representa como un par (a, b) donde a es la tripleta que representa a una ciudad y b es la tripleta que representa a otra.

En el algoritmo divideAndConquerTSP se consideran cuatro casos de acuerdo a la cantidad n de ciudades en el arreglo P. Si $n = 1$, $n = 2$ o $n = 3$, se emplean las funciones cicloUnaCiudad, cicloDosCiudades o cicloTresCiudaes, respectivamente, para obtener un tour que recorra las n ciudades en orden. Por otro lado, si $n > 4$ se procede a emplear los algoritmos obtenerParticiones y combinarCiclos para dividir y conquistar el problema.

En el algoritmo obtenerParticiones se emplean rectángulos, dados por arreglos de pares de coordenadas (pares de Doubles), para poder separar el conjunto P de ciudades en dos particiones más pequeñas que permitan resolver el problema.

1. Primero se emplea la función obtenerRectángulo para obtener 4 coordenadas (pares de doubles) que delimitan un rectángulo que contiene a todas las ciudades de P. Dichas coordenadas se encuentran en el siguiente orden: esquina inferior izquierda, esquina inferior derecha, esquina superior derecha, esquina superior izquierda.
2. Posteriormente se determina un eje de corte de acuerdo a las dimensiones del primer rectángulo y este se emplea para dividir dicho rectángulo en dos partes, trazando una recta perpendicular a su lado más largo. Dicha recta se traza en un punto determinado por la función obtenerPuntoDeCorte, la cual ordena P de acuerdo a las coordenadas del eje de corte y empleando una modificación del algoritmo randomized quicksort, para luego seleccionar su punto medio como punto de corte.

3. Luego de trazar dicha recta por medio de la función aplicarCorte, se obtienen dos rectángulos (izquierdo y derecho o inferior y superior) los cuales vienen dados por arreglos de pares con sus 4 coordenadas (en el mismo orden que el primer rectángulo) y una coordenada adicional que indica cuál rectángulo es (el izquierdo o el derecho) y respecto a cuál eje de corte fue obtenido. En esta última coordenada se emplea 1.0 y 0.0 para referirse al primer y al segundo rectángulo, respectivamente; y 2.0 y 3.0 para referirse a los ejes X y Y, respectivamente.

4. Posteriormente se emplea la función obtenerPuntosRectangulo para separar P en dos particiones (arreglos de tripletas) dadas por los rectángulo obtenidos. En este caso los puntos del lado común a ambos rectángulos (lado de corte) se almacenan en el primero (izquierdo o inferior, según corresponda).

5. En caso de que alguna de las particiones quedara vacía, se repite el proceso anterior tomando como eje de corte el opuesto al primero. Y si, por segunda vez, alguna partición queda vacía, se realiza el corte por el punto medio de uno de los ejes del rectángulo original y se repite lo anterior.

En el algoritmo combinarCiclos se reciben dos ciclos obtenidos del divideAndConquerTSP y se mezclan en un sólo ciclo considerando la forma más óptima de unir los lados para tener la menor distancia posible.

1. Primero se verifica si alguno de los ciclos recibidos es vacío, en cuyo caso retorna el otro ciclo.

2. Posteriormente se verifica lado por lado en ambos ciclos para determinar cuál es la combinación óptima para unirlos. Con esto se señala qué lados deben removerse y qué lados agregarse al momento de la unión.

3. Una vez determinado cuales lados deben retirarse y cuales añadirse, se procede a remover los lados de los dos ciclos a través de la función *remover*, y posteriormente se combinan ambos ciclos con la función *tour*, ordenando los lados a agregar de tal manera que se siga la secuencia de un ciclo.

Finalmente, en el algoritmo divideAndConquerAndLocalSearchTSP se emplea el procedimiento de optimización 2-opt para mejorar el resultado proporcionado por divideAndConquerTSP. Dicha optimización consiste en intercambiar los lados de un tour hasta obtener una distancia más óptima y para implementarla se emplea la función swap2OPT, la cual lleva a cabo el cambio e invierte la sección correspondiente en el tour para mantener el recorrido ordenado.

Se tiene que el algoritmo implementado no es completamente funcional pues presenta problemas al momento de realizar las particiones en el algoritmo 2 y al momento de combinar los ciclos en el algoritmo 3. Sin embargo, representa una base para poder desarrollar un mejor algoritmo en el futuro y muestra la utiliza del optimizador 2opt.

También cabe mencionar que en el algoritmo implementado se pueden obtener resultados variables en las distancias, esto es debido a la forma en que fue codificado el optimizador 2opt.

3. Resultados experimentales

A continuación se presentan los resultados de las comparaciones entre el algoritmo implementado y los resultados presentados en TSPLIB.

El estudio experimental se llevó a cabo en un computador Intel® Core™ i5-2450M CPU @ 2.50GHz × 4, con 8Gb de RAM y sistema operativo Ubuntu 20.04.6 LTS. Además, se empleó el lenguaje de programación Kotlin en su versión 1.8.21 y Java Virtual Machine JVM, versión 11.0.19.

Nombre de la instancia	**Distancia obtenida**	**% Desviación del valor óptimo**
berlin52	8108	
bier127	137110	
brd14051	55549	
ch130	6856	
ch150	7226	
d198	16753	
d493	39369	
657	56404	
d1291	60319	
1665	74891	
2103	103020	
15112	1869131	
eil51	474	
eil76	579	

Nombre del resolvedor	Distancia total obtenida	% Desviación del valor óptimo total
Divide-and-conquer		
Divide-and-conquer + 2opt		

4. Conclusiones

Es importante destacar que la realización del presente proyecto y el estudio del problema del viaje permitieron detallar y comprender la utilidad de los algoritmos de recursión y divide-and-conquer al momento de trabajar con grandes cantidades de datos. Además, se apreció como la técnica de subdividir una instancia en problemas más pequeños proporciona un esquema más cómodo y útil para implementar algoritmos.