Universidad Simón Bolívar
Proyecto 1

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

Autor:

Miguel Salomon
19-10274

Profesor:

Guillermo Palma

Caracas, 18 de junio 2023

1) Diseño de la solución

La implementación de los algoritmos se realizó en el lenguaje de programación Kotlin. La primera decisión de diseño tomada fue con qué estructuras se representan los elementos del algoritmo dado, y de esta forma, el marco de trabajo escogido fue el siguiente:

- Ciudades: Representadas como pares de números reales, con la estructura Pair, para la cual se usó el alias de Point.
- Particiones: Representadas como arreglos de ciudades, usando la estructura Array, para la cual se definió el alias de CityArray.
- Lados: Representados como pares de ciudades, mediante la estructura Pair.
- Ciclos: Representados como arreglos de lados, usando la estructura Array>, para la cual se definió el alias de Cycle.
- Rectángulos: Se decidió representarlos usando sólo dos coordenadas, aquellas correspondientes a las esquinas inferior izquierda y superior derecha que delimitan el área del mismo. La estructura usada fue Pair, a la cual se le dio el alias de Rectangle. Cabe destacar que aunque la estructura usada fue la misma que para los lados, en el código fuente se distingue entre Pair como un lado y Rectangle con un rectángulo.

Para el algoritmo obtenerPuntoDeCorte se realizó una modificación del algoritmo QuickSort, el cual recibe una matriz de ciudades y un carácter que indica el eje según el cual se deben ordenar los pares de la entrada. Después de varias pruebas, se decidió limitar el tamaño de las secuencias a ordenar a 16 usando el algoritmo de InsertionSort. Esta limitación mostró mejores resultados.

La implementación del algoritmo combinarCiclos presentó algunos problemas. Uno de estos fue mantener los lados en el orden correcto durante la combinación, de manera que los lados adyacentes en el ciclo resultante siempre compartieran una de sus ciudades. Se crearon algoritmos para lograr esto en un solo procedimiento, agregando los lados al ciclo3 mientras se mantenía el orden, pero fallaba en varias instancias, por lo que se decidió dividir el problema de combinar en dos partes. Primero, se concatenaron el ciclo1 y el ciclo2, reemplazando los lados que debían eliminarse con los lados que se agregarían. Luego, se permuta el arreglo ciclo3 para obtener el ciclo deseado, en el que cada lado compartiera una ciudad con su adyacente en el arreglo.

Para lograr la permutación del ciclo resultante de esta manera, se implementó un algoritmo basado en Selection Sort. En este algoritmo, para cada lado del arreglo, se busca el lado que comparte una ciudad con él y se intercambia el elemento siguiente por el correspondiente.

Se decidió trabajar con distancias en números reales de doble precisión. Esta decisión se tomó para minimizar la ganancia absoluta al reemplazar un par de lados antiguos por uno nuevo. Los redondeos podrían causar que se ignore una solución mejor que solo difiere en unos pocos decimales de una solución anterior, lo que podría mejorar más del 1% la calidad de las soluciones. Debido a esta mejora significativa, se mantuvo esta decisión en la versión final del algoritmo.

En la implementación del algoritmo aplicarCorte, el algoritmo obtenerPuntosRectángulo consistió en agregar todos los puntos que se encuentran dentro del área delimitada por el rectángulo dado, incluyendo todas sus aristas. Dado que se utilizó la estructura de datos Array para implementar las particiones, las cuales tienen un tamaño fijo, se buscaron los índices "a" y "b" de la partición de entrada que contienen todas las ciudades dentro del rectángulo. El arreglo de entrada siempre estará ordenado con respecto al eje de corte, lo que significa que todos los puntos dentro de un rectángulo siempre conformarán una sección continua del CityArray de entrada.

Además, el programa también incluye la implementación de las funciones extraerDatos y escribirTSP. La función extraerDatos convierte los datos de un archivo en formato TSPLIB al arreglo de ciudades correspondiente, mientras que la función escribirTSP es responsable de escribir los resultados en un archivo de salida. Esta última función mapea cada ciudad según su posición en el arreglo inicial.

2) Resultados experimentales

Los siguientes son los detalles de la máquina y el entorno donde se ejecutaron los algoritmos:

-Sistema operativo: Ubuntu 22.04.
-Procesador: AMD Ryzen 5 5600H 3.30GHz.
-Memoria RAM: 16 GB
-Compilador: kotlinc-jvm 1.8.21 (JRE 17.0.6+10).
-Entorno de ejecución: OpenJDK 64-bit Server VM Temurin-17.0.6+10

A continuación se presenta un tabla con todos los resultados obtenidos de la ejecución del programa sobre las instancias de prueba contenidas en estp_instancias.tar.xz y su desviación con respecto a la solución óptima:

Nombre de la instancia	Distancia obtenida	%Desv. valor óptimo
berlin52	8341	17.5
bier127	128892	21.15
brd14051	565609	20.48
ch130	6259	16.48
ch150	6910	19.45
d1291	61690	29.86
d15112	1908216	21.3
d1655	69764	19.74
d198	16750	11.82
d2103	97745	33.38
d493	36826	18.9
d657	55533	24.52
eil101	639	10.65
eil51	412	9.15
eil76	547	9.29
fl1400	25581	37.45
fl1577	30056	41.71
fl3795	37933	38.05
fl417	1547	37.16
fnl4461	193835	20.3
gil262	2542	18.63
kroA100	23635	17.89
kroA150	29889	19.6
kroA200	33578	19.94
kroB100	22188	12.51
kroB150	29102	20.75

kroC100	23246	18.63
rkoD100	23683	16.43
rkoE100	23070	14.26
lin105	15296	18.08
lin318	47114	21.84
nrw1379	60614	19.7
p654	45922	36.79
pcb1173	62515	23.52
pcb3038	152296	21.91
pcb442	54126	16.58
pr1002	297549	25.28
pr107	48726	14.27
pr124	66134	18.88
pr136	91016	11.17
pr144	62274	7.14
pr152	84747	20.54
pr226	94644	22.91
pr2392	434163	27.34
pr264	56239	19.75
pr299	53209	23.05
pr439	122127	23.69
pr76	113216	17.25
rat195	2516	16.92
rat575	6928	17.38
rat783	9710	19.11
rat99	1313	14.12
rd100	7964	8.22
rd400	16713	19.04
rl11849	1179874	27.78
rl1304	316868	34.58
rl1323	335179	32.91

rl1889	392268	35.12
rl5915	692800	31.48
rl5934	689396	32.41
st70	631	20.44
ts225	135766	10.91
tsp225	4190	16.74
u1060	250264	24.35
u1432	156816	15.69
u159	4592	18.98
u1817	65503	27.3
u2152	71426	21.98
u2319	213689	4.25
u574	40840	22.56
u724	46723	21.83
vm1084	282705	26.41
vm1748	401527	31.29

Podemos ver que el mejor resultado no es obtenido de la instancia m´as peque˜na, y la instancia con m´as ciudades (brd14051) obtiene una solución cuya desviación ronda el promedio de desviaciones señalado anteriormente (20.48 %).

Se puede ver de instancias como u2319 y rd100, la cuales conforman el mejor y el tercer mejor resultado de todos los casos de prueba, respectivamente, que como se espera, son las instancias distribuidas de forma más equilibrada en el plano las que logran obtener mejores resultados.

Ahora presentamos una tabla con los datos consolidados:

Nombre del resolvedor	Distancia total obtenida	%Desv. Total obtenido
Divide-and-Conquer	12166355	496232,5
Divide-and-Conquer + 2opt	10752146	496232,5

Con esta tabla podemos ver que el resolvedor Divide-and-Conquer + 2opt efectivamente arroja una distancia menos a la del resolvedor Divide-and-Conquer tal como se esperaba con el mismo porcentaje de desviación total. sin embargo lo realiza en un tiempo significativamente mayor para las instancias de mayor cantidad de “ciudades”.

Conclusiones y lecciones aprendidas:

Podemos concluir que la técnica de Divide-and-conquer es útil para diseñar algoritmos y puede proporcionar enfoques interesantes para abordar problemas conocidos. Sin embargo, es crucial tener cuidado al dividir el problema y combinar las soluciones de los subproblemas más pequeños para obtener la solución óptima. Aunque pueda parecer que ciertos algoritmos, como "obtenerPuntosRectangulo" o "obtenerPuntoDeCorte", son innecesarios, en realidad, la calidad de las soluciones encontradas depende de cómo se divide el arreglo de entrada en subproblemas cada vez más pequeños.

Los algoritmos de ordenamiento tienen un alcance mucho más amplio que simplemente ordenar secuencias de números enteros o reales. Comprender cómo funcionan estos algoritmos permite implementarlos como parte de soluciones en las que se requiere ordenar datos de una forma diferente a la habitual.

En general, estos algoritmos de ordenamiento se pueden implementar siempre que los elementos a ordenar sean comparables.

Seleccionar las estructuras de datos adecuadas es crucial al trabajar con conjuntos de datos en un proyecto. Cada estructura de datos tiene un desempeño óptimo para ciertos propósitos, lo que afectará la facilidad de manejo de las implementaciones y la eficiencia del programa final. Por lo tanto, es importante elegir cuidadosamente las estructuras de datos que se utilizarán en el proyecto para lograr los mejores resultados posibles.