Traveling Salesman Problem - Genetic Algorithm Solution

Sebastián Mederos sebastian.mederos@uc.edu.py
Sistemas Paralelos y Distribuidos, Facultad de Ciencias y Tecnología
Universidad Católica Nuestra Señora de la Asunción
Asunción - Paraguay

Abstract

El problema del agente viajero (TSP) es uno de los más estudiados en el campo de la optimización, como también en la matemática computacional. El mismo consiste en el recorrido de lugares o nodos (para entregar o recoger mercancías) con el fin de resolver problemas que impidan minimizar o maximizar algún objetivo (tiempo o costos). En el presente artículo se presenta una descripción general y una explicación de este problema y además, una implementación de la heurística de GA (Genetic Algorithm) paralelizada para determinar y analizar una solución con mayor rapidez y eficiencia.

Introducción y justificación

El problema del agente viajero TSP responde a la siguiente pregunta: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen? Este es un problema considerado NP-Hard en optimización combinatoria, muy importante en la investigación de operaciones y en la ciencia de la computación.

Fue formulado por primera vez en 1930 y es uno de los problemas de optimización más estudiados. Es usado como prueba para muchos métodos de optimización. Aunque el problema es computacionalmente complejo, son conocidos una gran cantidad de heurísticas y métodos exactos, de manera que puede ser resuelto. [1]

El TSP tiene diversas aplicaciones, tales como: la planificación, en la fabricación de circuitos electrónicos o secuencia de ADN. En estas aplicaciones, el concepto de "ciudad" representa, por ejemplo: clientes, puntos de soldadura o fragmentos de ADN y el concepto de "distancia" representa el tiempo de viaje o costo, o una medida de similitud entre los fragmentos de ADN.

Existen varios métodos empleados para resolver el problema del TSP. En este caso, la heurística utilizada es Genetic Algorithm (Algoritmo Genético - GA).

Los algoritmos genéticos o GA son métodos adaptativos que brindan soluciones a problemas de búsqueda y optimización. Básicamente, imitan el comportamiento o proceso evolutivo de los seres vivos. Es decir, con el transcurso de las generaciones, las poblaciones evolucionan en la naturaleza conforme a los principios de la selección natural y la supervivencia de los más fuertes. Por tal motivo, este tipo de algoritmos se consideran aptos para desarrollar soluciones a problemas del mundo real.
Una de las razones de utilizar los algoritmos genéticos es que tienden a ser paralelizables, en este trabajo estaremos presentando dicha técnica. [2]

Objetivos

El trabajo persigue comprender el proceso de paralelizar el Traveling Salesman Problem haciendo uso de una heurística en particular, para un sistema con memoria compartida usando OpenMP.

Asimismo, realizar la evaluación de desempeño del algoritmo utilizando varios tamaños de datos de entrada, proveyendo como pruebas las siguientes métricas:

- Gráficos de las curvas de Speedup y Eficiencia
- Tiempo de ejecución

Metodología

El método elegido consiste en un esquema de paralelización para el algoritmo heurístico planteado Genetic Algorithm (Algoritmo Genético - GA).

1. Genetic Algorithm (Algoritmo Genético)

Para resolver este problema de optimización se hará uso de los algoritmos genéticos. Un algoritmo genético se encarga de crear una población de soluciones a un problema y aplicar una serie de operadores que permitirán evolucionar esas soluciones, a una nueva población posiblemente mejor.
Las poblaciones son un conjunto de cromosomas, en donde cada cromosoma tiene un conjunto de genes, por lo general estos cromosomas son representados mediante cadenas binarias. Sin embargo, para el problema del agente viajero no resulta conveniente aplicar una representación de ese tipo, ya que cada ciudad esta representa con una coordenada (x,y) que es su ubicación, por lo que lo más adecuado y eficaz es trabajar con cadenas de números enteros.
De esta manera, simplemente bastará con enumerar las ciudades a visitar por ejemplo 1 hasta n ciudades. Por lo tanto, la cadena (cromosoma o individuo) contendrá una posible permutación que va de 1 hasta n la cual simboliza un posible recorrido, con sus coordenadas (x,y) respectivamente. Las demás características para el proceso evolutivo son las siguientes:

Población inicial: la población inicial se genera de manera aleatoria de acuerdo a las coordenadas (x,y) de las ciudades.

Fitness: el costo del recorrido de las ciudades de acuerdo al orden de los genes de nuestro cromosoma, el fitness con un valor menor será el mejor.

Selección: para la selección de los individuos para el cruce se usará la selección por torneo, en donde se selecciona k-individuos de la población, en donde se elegirá el al individuo con mejor fitness de los k-individuos seleccionados.

Cruce: como los genes del cromosoma no pueden repetirse se utiliza el cruce de dos puntos, para este caso se hará uso de PX (Position-based Crossover) que es una variación del OX(order crossover).

Mutación: la técnica consiste en seleccionar dos números aleatorios entre 1 y el tamaño del cromosoma, estos dos valores serán dos posiciones de la cadena del individuo. Posteriormente, se intercambia el contenido de dichas posiciones.

Condición de Fin: como desconocemos la solución óptima, se repetirá el proceso evolutivo hasta la generación N, en este caso 99. [3]

Pasos para ejecturar el proyecto

1. Abrir una ventana de terminal en la carpeta principal del proyecto. [4]
2. Compilar el proyecto con el comando:
   ```
   gcc -o GA par_main.c -fopenmp -l
   ```
3. Ejecutar el proyecto con el comando:
   ```
   ./tsp_NUM numero_de_hilos
   ```

Observaciones

Se puede observar que a medidas que ejecutamos con más hilos mejora el tiempo de ejecución tal y como se esperaba, como también las soluciones se acercan más a las soluciones reales, esto porque a medida que tenemos más hilos obtenemos muchas más diferentes soluciones, ya que nuestra estrategia de paralelización se basa en cada hilo genere una secuencia de descendiente y como es aleatorio las selecciones uno de los hilos puede evitar los mínimos de las rutas y acercarse mas a la solución real, es decir, al mínimo óptimo del recorrido.

Resultados

Se han realizado pruebas para evaluar el desempeño del algoritmo serial y paralelo del GA en términos de tiempo de CPU y precisión en el resultado para las ciudades. Ambas versiones están codificadas en lenguaje C++ y para la paralelización se ha utilizado la librería OpenMP. Las pruebas fueron realizadas en una máquina con sistema operativo Centos 7, procesador i5-8300H @ 2,30 GHz, 8 Cores/16 Threads y, con 8GB de memoria RAM, utilizando cuatro, ocho y dieciséis threads para la prueba. Se utilizaron tres tipos de datos realizando la definición de manera global para los conjuntos de ciudades 300, 512 y 700 nodos. En la siguiente tabla se muestran los resultados que se obtuvieron con ambas versiones, el tiempo está medido en segundos, siendo P el número de hilos utilizado en la ejecución de la solución paralela:

P=4	GA Serial		GA Paralelo		Speedup	Eficiencia
Conj. de ciudades	Tiempo	Fitness	Tiempo	Fitness		
300	36.018248s	470155787	34.558712s	30601875	1.042233518	0.2605583
512	96.763499s	1026680827	96.282712s	989113706	1.004993492	0.2512483
700	205.216391s	2108577922	206.146201s	2349184593	0.995489560	0.2488723

Figura 1: Análisis y comparación de tiempos con 4 threads

Con la ejecución en modo paralelo con 4 hilos, se obtiene una reducción del tiempo de ejecución como también una solución más próxima en comparación a la solución conocida y menor a la solución con ejecución serial, con excepción del tercer conjunto de ciudades.

P=8	GA Serial		GA Paralelo		Speedup	Eficiencia
Conj. de ciudades	Tiempo	Fitness	Tiempo	Fitness		
300	36.018248s	470155787	32.080465s	32371844	1.122747067	0.140343383
512	96.763499s	1026680827	83.700408s	961448933	1.156069621	0.144508702
700	205.216391s	2108577922	203.147333s	1965718904	1.0101850118	0.126273126

Figura 2: Análisis y comparación de tiempos con 8 threads

Con la ejecución en modo paralelo con 8 hilos, se obtiene una reducción del tiempo de ejecución como también una solución más próxima en comparación a la solución conocida y menor a la solución con ejecución serial.

P=16	GA Serial		GA Paralelo		Speedup	Eficiencia
Conj. de ciudades	Tiempo	Fitness	Tiempo	Fitness		
300	36.018248s	470155787	26.029488s	27041517	1.383747847	0.08648424
512	96.763499s	1026680827	79.102817s	911601875	1.223262364	0.07645389
700	205.216391s	2108577922	198.74063s	1481705614	1.0325839814	0.06453649

Figura 3: Análisis y comparación de tiempos con 16 threads

Con la ejecución en modo paralelo con 16 hilos, se obtiene una reducción del tiempo de ejecución como también una solución más próxima en comparación a la solución conocida y menor a la solución con ejecución serial.

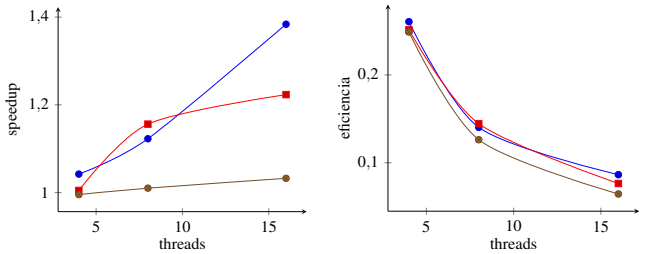

Figura 4: Curva gráfica del speedup y eficiencia

Speedup y eficiencia

Se logra observar en la figura 4 la curva de speedup, la cual no precisamente se asemeja a su ideal esperado, debido a las variaciones de procesos en tiempo de ejecución, en sentido que resulta imposible llevar un control total del rendimiento del procesador en un sistema operativo completo, que tiene múltiples tareas en simultáneo, y por momentos obtiene mejor rendimiento que en otros. Pero de forma general, puede apreciarse que a medida que se aumenta el número de hilos en el procesamiento mejora el speedup de la solución. Por otra parte, la eficiencia, por definición va tomando una curva en declive en relación al número de hilos utilizados.

Conclusión

En este artículo se ha propuesto un esquema de paralelización para el algoritmo de GA utilizando OpenMP y se ha evaluado utilizando conjuntos de datos con números de ciudades distintos de manera a ver el desempeño en tiempo de CPU así como la precisión en el resultado para las ciudades utilizadas. Se ha confirmado la consistencia relativa entre las versiones secuencial y paralela del algoritmo, se verificó que sus resultados no difieren en una cantidad significativa, y las pequeñas diferencias son debido al número de hilos utilizados, la principal divergencia se da por el número de ciudades.

Código Fuente del Proyecto

Enlace al Repositorio del Proyecto en GitHub

Referencias

[1] "Problema del viajante wikipedia [online]," https://es.wikipedia.org/wiki/Problema_del_viajante, año: 2021.

[2] P. A. Pétrowski, J.Dréo; E. Tailard, "Metaheuristics for hard optimization," *Springer*, 2005.

[3] T. . R. M. L. De Carvalho, R. L.; da Silva Almeida, "Introdução às metaheurísticas," *Portal de Livros da Editora, 1(18), Lv18.*, 2020.

[4] "Simple genetic algorithm for solving the travelling salesman problem (tsp). [online]," https://github.com/freetonik/gen_tsp, año: 2014.