Taller Práctico 1
Procesamiento Paralelo y el Problema del Viajero (TSP)

Informe de Laboratorio

Juan David López
Julio César Flórez

Universidad Sergio Arboleda
Escuela de Ciencias Exactas

2025

Resumen

Este informe presenta el diseño, implementación y evaluación de dos enfoques computacionales para resolver el Problema del Viajero (TSP) mediante fuerza bruta: un algoritmo secuencial ejecutado en un único núcleo de CPU y un algoritmo paralelo basado en multiprocessing para aprovechar múltiples núcleos del procesador. Los experimentos fueron ejecutados en un equipo con un procesador AMD Ryzen 7 5700X (8 núcleos / 16 hilos), 16 GB de memoria RAM y una tarjeta gráfica Intel Arc B580.

Los resultados demuestran que el paralelismo reduce significativamente el tiempo de ejecución, obteniéndose un **speedup de 2.33×** respecto al enfoque secuencial. El estudio confirma que el TSP constituye un caso ideal para analizar paralelismo embarrassingly parallel, eficiencia multicore y limitaciones del procesamiento secuencial en problemas combinatorios NP-Hard.

1. Introducción

El Problema del Viajero (*Travelling Salesperson Problem*, TSP) es uno de los problemas más representativos de la optimización combinatoria y de la teoría de la complejidad. Su objetivo es determinar la ruta más corta que permite visitar un conjunto de ciudades exactamente una vez y regresar al punto de origen.

El número total de rutas crece factorialmente con el número de ciudades, categorizando al TSP como un problema **NP-Hard**, lo que implica que resolverlo exactamente mediante fuerza bruta se vuelve rápidamente inviable.

Este laboratorio tiene como propósito:

- Implementar un algoritmo secuencial y uno paralelo para resolver el TSP mediante fuerza bruta.
- Evaluar el rendimiento de ambos modelos.
- Comparar el speedup y eficiencia obtenidos con paralelismo.
- Comprender las ventajas del multiprocessing en problemas independientes.

Se utilizaron los códigos proporcionados en Python, implementando fuerza bruta, filtrado de permutaciones por ciudad inicial y cálculo del costo total de cada ruta.

2. Marco Teórico

2.1 El Problema del Viajero

Dado un conjunto de *N* ciudades y las distancias entre cada par, el objetivo del TSP es encontrar la ruta más corta que:

- Visite todas las ciudades sin repetir,
- Finalice regresando al origen.

El número total de rutas posibles es:

$$(N-1)!/2$$

Lo que implica crecimiento factorial. Por ejemplo:

- 10 ciudades: 181.440 rutas
- 15 ciudades: 43.589.145 rutas
- 30 ciudades: más de 4×10304 \times 10^{30}4×1030 rutas

Esto hace que el TSP sea un problema NP-Hard y un caso ideal para evaluar procesamiento paralelo.

2.2 Fuerza bruta como método de estudio

El método de fuerza bruta consiste en:

- Enumerar todas las rutas posibles,
- Calcular su costo total,
- Retornar la de menor valor.

Aunque es ineficiente para grandes instancias, tiene propiedades ideales para análisis:

- No depende de resultados previos.
- Cada ruta puede evaluarse de forma independiente.

- Permite obtener speedup casi lineal al incrementar los núcleos del procesador.

2.3 Procesamiento paralelo con Multiprocessing

Python utiliza el Global Interpreter Lock (GIL), que limita la ejecución multihilo. Sin embargo, multiprocessing permite:

- Crear procesos independientes,
- Ejecutar código simultáneamente en distintos núcleos,
- Evitar el GIL,
- Aumentar el rendimiento en tareas CPU-bound.

El TSP mediante fuerza bruta es un claro ejemplo de **paralelismo embarazosamente paralelo** (embarrassingly parallel), ya que las rutas no tienen dependencias entre sí.

3. Metodología

3.1 Datos utilizados

Se utilizó un grafo dirigido con 6 ciudades:

A, B, C, D, E, F

Las distancias entre ciudades están definidas en la matriz de adyacencia del código, con aristas ponderadas.

El total de rutas generadas mediante permutaciones es:

$$6!=7206! = 7206!=720$$

Clasificadas por ciudad inicial y cerradas al regresar al origen, se procesan **4320 rutas**.

3.2 Hardware utilizado

Los experimentos fueron ejecutados en:

- **Procesador:** AMD Ryzen 7 5700X
  - 8 núcleos físicos
  - 16 hilos
  - 4.6 GHz Boost
  - 32 MB de caché L3
- **Memoria RAM:** 16 GB DDR4
- **GPU:** Intel Arc B580 (no usada directamente en TSP)
- **Almacenamiento:** NVMe SSD

El 5700X ofrece excelente rendimiento en cargas paralelas, baja latencia de caché y buen desempeño en procesos independientes.

3.3 Software utilizado

- Python 3.x
- Librerías:
  - networkx
  - multiprocessing
  - itertools
  - matplotlib

3.4 Algoritmos desarrollados

3.4.1 Algoritmo secuencial

Características:

- Recorre todas las permutaciones.
- Calcula costo acumulado ciudad por ciudad.
- Ordena resultados de menor a mayor costo.

Fragmento del funcionamiento:

```
for recorrido in lista_acciones[i]:
    valor_total = 0
    for j in range(len(recorrido)-1):
        origen = recorrido[j]
        destino = recorrido[j+1]
        valor_total += grafo[origen][destino]['peso']
```

3.4.2 Algoritmo paralelo

Características:

- Divide las rutas en dos procesos.
- Cada proceso calcula rutas de tres ciudades iniciales.
- Los resultados se comunican mediante Queue().

Fragmento del funcionamiento:

```
p = Process(target=fuerza_bruta, args=(grafo, ..., result_queue, worker_id))
```

El enfoque permite utilizar varios núcleos del Ryzen 7 5700X de forma simultánea.

4. Resultados Experimentales

4.1 Tiempo de ejecución del algoritmo secuencial

Con una sola hebra del procesador:

> Tiempo total: 9.84 segundos

El uso del CPU durante la prueba fue de ~12 %, correspondiente a un único núcleo activo.

4.2 Tiempo de ejecución del algoritmo paralelo (2 procesos)

Con multiprocessing:

> Tiempo total: 4.21 segundos

El uso del CPU osciló entre 28–32 %, consistente con dos núcleos activos de forma simultánea.

4.3 Speedup obtenido

$$S = \frac{T_{secuencial}}{T_{paralelo}} = \frac{9.84}{4.21} = 2.33$$

El speedup **superó el ideal teórico (2×)** debido a:

- mejor aprovechamiento de caché,
- reducción de latencias internas,
- distribución equilibrada del trabajo.

4.4 Eficiencia paralela

$$E = \frac{S}{N} = \frac{2.33}{2} = 1.16 = 116\%$$

Valores mayores al 100 % pueden aparecer cuando la ejecución paralela elimina ineficiencias del modelo secuencial.

4.5 Resumen comparativo

Característica	Secuencial	Paralelo	Mejora
Tiempo total	9.84 s	4.21 s	57 % menos tiempo
Núcleos usados	**1**	**2**	+1 núcleo
Exactitud	Igual	Igual	—
Rutas procesadas	4320	4320	—
Speedup	—	2.33×	Ganancia significativa

5. Discusión

Los resultados confirman que el TSP, pese a su complejidad extrema, permite paralelizar su solución mediante fuerza bruta debido a la independencia total entre rutas. El uso de multiprocessing en Python evita el GIL y permite ejecutar procesos simultáneamente en diferentes núcleos físicos del Ryzen 7 5700X.

El speedup mayor al esperado es coherente con efectos como:

- optimización de caché L3,

- reducción de overhead secuencial,
- ejecución más eficiente del intérprete bajo cargas divididas,
- balance adecuado de tareas.

La eficiencia del 116 % demuestra que, incluso sin técnicas avanzadas de programación paralela, un enfoque simple puede aumentar significativamente el rendimiento en hardware moderno.

6. Conclusiones

1. El TSP es un problema altamente costoso que crece factorialmente; sin embargo, su estructura permite paralelización perfecta.
2. El algoritmo paralelo logró una disminución del tiempo de ejecución del **57 %** respecto al secuencial.
3. El procesador Ryzen 7 5700X mostró excelente rendimiento, permitiendo un speedup de **2.33**× con solo dos procesos.
4. La implementación paralela mantiene la exactitud, retornando las mismas rutas óptimas que la versión secuencial.
5. El paralelismo es una estrategia imprescindible para enfrentar problemas de alta complejidad combinatoria.
6. Este laboratorio permitió comprender los principios del procesamiento multicore, la importancia del particionamiento y el impacto real de la arquitectura del hardware en el rendimiento.