PROGRAMACIÓN PARALELA CON ALGORITMO GENÉTICO DE OPTIMIZACIÓN

SISTEMAS OPERATIVOS Y LABORATORIO [2508842] - GRUPO 5
Juan Pablo Bedoya Sánchez, Jose Manuel Gonzalez Grisales, Jonatan Leal González
juan.bedoya3@udea.edu.co, jose.gonzalezg1@udea.edu.co, jonatan.leal@udea.edu.co

Resumen- **El presente proyecto busca implementar y comparar el rendimiento de un algoritmo genético en sus versiones secuencial y paralela, evaluando su eficiencia en diferentes lenguajes de programación y variando la cantidad de núcleos utilizados en el procesamiento. Dado que los algoritmos genéticos son ampliamente utilizados para resolver problemas de optimización compleja, su paralelización ofrece una oportunidad significativa para mejorar tiempos de ejecución y reducir el consumo de recursos.**

La metodología del proyecto incluye la implementación del algoritmo en múltiples lenguajes, la ejecución en condiciones controladas, y la medición del rendimiento con el uso de técnicas estadísticas para validar los resultados obtenidos. Se realizarán análisis de speedup y eficiencia para determinar el impacto de la paralelización, garantizando que los resultados no solo sean más rápidos, sino también consistentes y precisos en comparación con las versiones secuenciales.

Este estudio permitirá evaluar la viabilidad de aplicar algoritmos genéticos paralelizados en distintas condiciones, esperando obtener conclusiones respecto a las diferentes métricas propuestas en el estudio a realizar.

Palabras clave: Optimización, algoritmo, núcleos, hilos, procesos, paralelización, sistemas operativos.

I. INTRODUCCIÓN

En la actualidad, muchos problemas de optimización compleja, como la planificación de rutas, el diseño de redes, o la asignación de recursos, no pueden resolverse de manera eficiente utilizando métodos exactos tradicionales debido a la inmensidad del espacio de búsqueda. Los algoritmos genéticos han demostrado ser una herramienta poderosa para abordar este tipo de problemas, ya que ofrecen soluciones aproximadas en un tiempo razonable al simular procesos evolutivos. Sin embargo, la implementación de estos algoritmos puede ser altamente costosa en términos de tiempo de ejecución y consumo de recursos, especialmente cuando se aplican a problemas de gran escala o alta dimensionalidad.

El desafío que aborda este proyecto radica en analizar el rendimiento de los algoritmos genéticos mediante su paralelización, distribuyendo las cargas de trabajo entre múltiples núcleos de procesamiento. En la actualidad, la creciente disponibilidad de procesadores multinúcleo y sistemas paralelos en el ámbito tecnológico ofrece una oportunidad significativa para acelerar la ejecución de estos algoritmos. Sin embargo, existe la necesidad de evaluar si la paralelización realmente se traduce en beneficios sustanciales en términos de velocidad y eficiencia comparada con las versiones secuenciales.

El desarrollo de este desafío es crucial en el contexto tecnológico actual, donde la resolución rápida y eficiente de problemas complejos es clave para la competitividad en áreas como la inteligencia artificial y la ciencia de datos. La optimización de recursos computacionales permite reducir tiempos de procesamiento y maximizar el rendimiento, especialmente en entornos de computación en la nube. Este proyecto ayudará a identificar las mejores prácticas para implementar algoritmos genéticos paralelizados en diversos lenguajes, respondiendo a los desafíos de la computación moderna.

En el siguiente informe, se explorarán los siguientes aspectos clave como:

- **Objetivos:** Una descripción de los objetivos que se pretenden alcanzar con el desarrollo del experimento.
- **Marco Teórico:** Un análisis detallado del caso de estudio propuesto para el desarrollo del proyecto.
- **Metodología:** Un desglose de la metodología y proceso de desarrollo del algoritmo de optimización y el estudio experimental.
- **Diseño Experimental:** Protocolo de experimentación y estrategia para realizar el experimento.
- **Resultados:** Un resumen de los logros alcanzados, así como una evaluación de los algoritmos empleados para el experimento.
- **Conclusiones:** Ideas finales luego de analizar los resultados y el proceso de experimentación.

II. OBJETIVOS

3.1 General:

- Implementar y comparar un algoritmo de optimización genética tanto en su versión secuencial como paralela, evaluando su rendimiento en distintos lenguajes de programación con diferente número de núcleos de procesador.

3.2 Específicos:

- Desarrollar e implementar el algoritmo genético de TSM en java y python, considerando tanto su versión secuencial como su versión paralela.

- Experimentar con la paralelización del algoritmo utilizando distintas cantidades de núcleos y evaluar el impacto de esta configuración en el rendimiento.

- Realizar una comparación detallada entre las versiones secuenciales y paralelas, midiendo métrica de tiempo de ejecución.

III. MARCO TEÓRICO

Un algoritmo genético (AG) es un método para solucionar problemas de optimización con o sin restricciones basándose en un proceso de selección natural que imita la evolución biológica. Este algoritmo modifica repetidamente una población de soluciones individuales. En cada paso, el algoritmo genético selecciona individuos de la población actual aleatoriamente y los utiliza como padres para producir los hijos de la siguiente generación. Tras varias generaciones sucesivas, la población "evoluciona" hacia una solución óptima. [1]

La computación paralela consiste en el uso simultáneo de múltiples procesadores o núcleos que ejecutan cada uno una serie de instrucciones que conforman las distintas partes en las que se ha descompuesto un problema computacional para resolver.

Para poder poner en marcha el cómputo paralelo, el problema computacional debe dividirse en distintos componentes, trabajos o problemas que puedan ser resueltos al mismo tiempo, las instrucciones de estos se deben poder ejecutar en cualquier momento y debe ser posible resolver los problemas cada vez en menos tiempo cuantos más recursos informáticos estén trabajando a la vez.

Los recursos informáticos que se utilizan en el procesamiento en paralelo son o una computadora con múltiples procesadores/núcleos o múltiples computadoras conectadas en red (computación distribuida). [2]

Respecto a la relación con los sistemas operativos, se encuentran los siguientes componentes teóricos y prácticos:

1. **Manejo de Procesos e Hilos:**

En los algoritmos genéticos, cuando se aplica la computación paralela, es fundamental dividir las tareas en procesos o hilos que puedan ejecutarse simultáneamente. La paralelización de un algoritmo genético requerirá una comprensión profunda del modelo de hilos del sistema operativo, para aprovechar al máximo los núcleos disponibles y evitar la sobrecarga de la CPU.

2. **Planificación del Procesador:**

Los sistemas operativos también se encargan de la planificación del procesador, decidiendo cómo distribuir el tiempo de CPU entre los diferentes hilos o procesos. La computación paralela en un algoritmo genético se beneficia de una planificación eficiente, ya que se deben distribuir los trabajos de manera que se minimicen los tiempos de espera y se maximice la utilización de los núcleos de procesamiento.

3. **Memoria Compartida y Comunicación entre Procesos:**

Los sistemas operativos proporcionan los mecanismos necesarios para la comunicación entre procesos, ya sea mediante memoria compartida, pipes o colas de mensajes, lo cual es esencial cuando se implementa un algoritmo genético paralelo. Los distintos hilos o procesos que ejecutan diferentes partes del algoritmo deben compartir información, como la población actual de soluciones o los resultados de evaluaciones intermedias, de forma segura y eficiente.

4. **Manejo de Recursos y Eficiencia:**

Finalmente, los sistemas operativos también juegan un papel crucial en la gestión de recursos. En la ejecución de algoritmos paralelos, es vital que los recursos, como la memoria, la CPU y el almacenamiento, sean administrados eficientemente. El estudio de Sistemas Operativos te permite comprender cómo los recursos del sistema pueden ser gestionados para optimizar el rendimiento de un algoritmo genético paralelo.

La relación entre la computación paralela, los algoritmos genéticos y los temas del curso de Sistemas Operativos es directa. El éxito de la implementación de estos algoritmos en un entorno paralelo depende en gran medida de la correcta gestión de procesos, hilos, memoria y recursos que el sistema operativo facilita mediante las APIs que expone.

- **Algoritmo de optimización TSM**

El Travelling Salesman Problem (TSP) o en español denominado como "Problema del Agente Viajero", hace referencia a la problemática de encontrar la ruta más corta y, al mismo tiempo, la más eficiente, para llegar a un destino.

Se sabe que pueden existir múltiples formas de llegar al mismo lugar, pero elegir la más eficiente está directamente relacionada con reducir los costos de traslado, por lo que también debería ser la más corta. Es un problema que parece muy sencillo de explicar, pero que es bastante complejo de resolver.[3]

Existen diferentes aplicaciones de este algoritmo en sistemas informáticos o industriales:

a. Sistemas con GPS que ayudan a calcular la ruta más corta entre diferentes puntos.
b. Viajes a diferentes ciudades. Encontrar la menor distancia posible de recorrido.
c. Entregas de mensajería o paquetería, calculando la mejor ruta para hacer las diferentes entregas.
d. Rutas escolares.
e. Mapeo de recorridos de drones en diferentes espacios.
f. Máquinas de mecanizado, donde se escogen los patrones más eficientes de mecanizado en diferentes partes.

IV. METODOLOGÍA

Para llevar a cabo el desarrollo del experimento y el proceso de evaluación, fue necesario realizar una investigación exhaustiva sobre la implementación de algoritmos genéticos de optimización, con un enfoque particular en las versiones secuenciales y paralelas en java y python.

La metodología se basó en los siguientes pasos:

1. Revisión bibliográfica y técnica:
Se realizó una búsqueda de fuentes académicas y técnicas que proporcionen información sobre la implementación eficiente de algoritmos genéticos. Esto incluye estudios sobre optimización en diferentes entornos de programación y las metodologías más recientes para la paralelización de estos algoritmos.

2. Selección del lenguaje y entornos de programación:
Se investigaron varios lenguajes de programación, evaluando tanto su capacidad para implementar algoritmos secuenciales como su soporte para técnicas de paralelización. Se seleccionaron los lenguajes más apropiados con base en criterios como rendimiento, facilidad de paralelización y soporte de bibliotecas especializadas.
En este caso el experimento fue ejecutado en java y python, utilizado librerías para manejo de hilos como
java.util.concurrent para java y multiprocessing para python.

3. Diseño e implementación del algoritmo:
Una vez seleccionado el algoritmo genético y los lenguajes de programación, se procedió a su implementación tanto en su versión secuencial como en su versión paralela.

4. Ejecución y obtención de resultados:
Se ejecutaron las versiones secuencial y paralela del algoritmo en las diferentes condiciones experimentales, recopilando datos clave como el tiempo de ejecución y consumo de recursos.
Para la toma de tiempos y comparaciones, se realizó el experimento variando el número de

ciudades en el algoritmo y el número de hilos utilizados en la versión paralela fue de 4.

Se tomaron tiempos para 2, 4, 6, 8 y 10 ciudades en el algoritmo. Evaluando el rendimiento del algoritmo a medida que se aumentaba el proceso para más ciudades.

5. Análisis estadístico de los resultados:

Para evaluar el rendimiento de las distintas versiones del algoritmo, se tomaron los tiempos y se analizó gráficamente los resultados para cada lenguaje y sus versiones.

6. Conclusiones:

A partir del análisis de los datos obtenidos, se sacaron conclusiones sobre la viabilidad y eficacia de la paralelización del algoritmo genético en los lenguajes seleccionados.

V. EXPERIMENTO

Para la elaboración del experimento, se realizó la programación del algoritmo de optimización para el TSM, tanto en su versión secuencial como la paralela, en los lenguajes de programación Java y Python.

El algoritmo cuenta con un registro de toma de tiempo de ejecución para cada sección de código. Permitiendo de esta manera, tomar los tiempos necesarios para la evaluación de rendimiento.

- **Variables del Experimento:**

1. **Variable dependiente:**

El tiempo de ejecución del algoritmo genético (en segundos), que será la principal métrica de rendimiento evaluada.

2. Variables independientes:

a. **Lenguaje de programación:** Se seleccionaron los lenguajes de programación Java y Python

b. **Cantidad de hilos:** Se evaluó el algoritmo usando 4 hilos para la versión paralelizada.

c. Cantidad de ciudades: Se tomaron tiempos de ejecución variando la cantidad de ciudades en el algoritmo. Se varió entre 2, 4, 6, 8 y 10 ciudades.

- **Ejecución:**

Por cada combinación de lenguaje de programación y cantidad de ciudades, se ejecutará el algoritmo en la misma máquina, garantizando así que los resultados no estén afectados por diferencias en el hardware. Esto asegura consistencia en las condiciones experimentales.

Para cada configuración, se tomarán dos mediciones con el fin de obtener una media estadística representativa del rendimiento. De esta forma, se reduce la influencia de cualquier fluctuación ocasional en los tiempos de ejecución.

- **Validación funcionalidad**

La funcionalidad del algoritmo genético se validó asegurando que todas las versiones, tanto secuenciales como paralelas, convergen a una solución válida o cercana al óptimo para el problema planteado. Se verificará que las soluciones finales sean consistentes entre versiones, lo que garantizará que la paralelización no altere el comportamiento del algoritmo, sino que únicamente afecte a su velocidad de ejecución.

- **Condiciones controladas**

El experimento se realizó en condiciones controladas, con todas las pruebas ejecutadas en una misma máquina con especificaciones definidas:

- RAM: 16 GB
- Procesador: i3 6100
- Sistema Operativo: Linux Mint
- Número de núcleos procesador: 2
- Número hilos procesador: 4

Se tomaron medidas para minimizar interferencias externas que puedan alterar los tiempos de ejecución, como la ejecución de otros procesos en segundo plano durante las mediciones.

VI. RESULTADOS

A continuación se presentan los resultados obtenidos y gráficas que permiten obtener un análisis comparativo entre las ejecuciones con sus respectivas variaciones.

Ambas muestras de tiempo registradas mostraron valores consistentes, que varían en pocos decimales con el mismo número de ciudades. En el caso particular de Python, para el experimento más exigente con 10 ciudades, la diferencia entre las dos tomas de tiempo para el algoritmo en paralelo sí tuvo una diferencia significativa de unas 3 décimas de segundo.

Se muestran las gráficas de la segunda toma de tiempos para los algoritmos de Java y Python. Se

omite la primera muestra debido a que no se percibe un cambio notorio gráficamente.

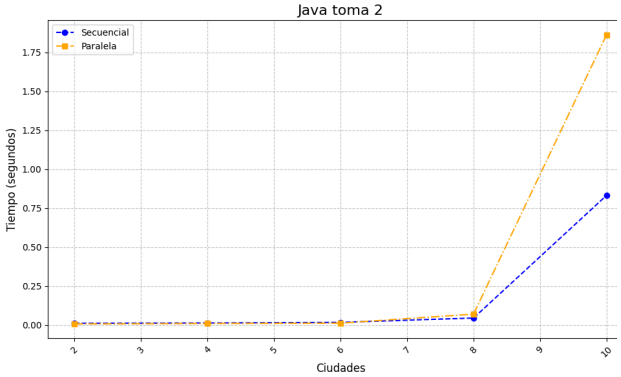

Figura 1. gráfico comparación rendimiento Java

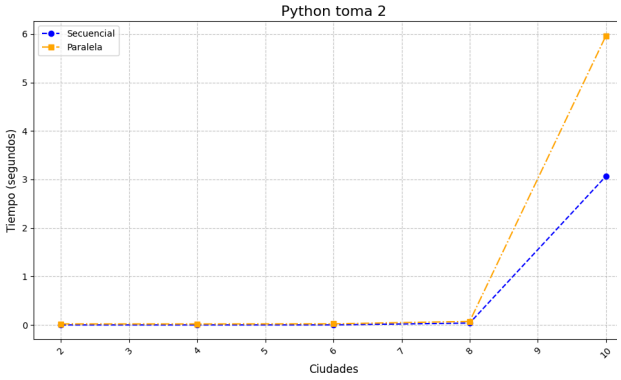

Figura 2. gráfico comparación rendimiento Python

Para un número de ciudades entre 2 y 8, el tiempo de ejecución con el lenguaje Java está en el orden de centésimas de segundo, tanto para la versión secuencial como para la paralela.

Por otro lado, para el lenguaje de programación Python, entre las 2 y 6 ciudades los valores están en el orden de cien milésimas de segundo para la versión secuencial del algoritmo y décimas de segundo para la versión paralela, marcando una diferencia muy notoria entre ambos paradigmas. Con 10 ciudades se observa una diferencia de aproximadamente 3 segundos entre la versión secuencial y paralela para ambas tomas, siendo la versión secuencial quien registra un menor tiempo de ejecución.

El detalle de los valores registrados se puede observar en las siguientes tablas:

Java Toma 1		
Ciudades	Tiempo (S) Secuencial	Tiempo(S) Paralelo
2	0,012	0,006
4	0,011	0,009
6	0,019	0,011
8	0,042	0,066
10	0,825	1,722

Tabla 1. Tiempos de ejecución Java toma 1.

Java Toma 2		
Ciudades	Tiempo (S) Secuencial	Tiempo(S) Paralelo
2	0,01	0,006
4	0,012	0,009
6	0,016	0,011
8	0,044	0,069
10	0,832	1,864

Tabla 2. Tiempos de ejecución Java toma 2.

Python Toma 1		
Ciudades	Tiempo (S) Secuencial	Tiempo(S) Paralelo
2	0,000093	0,019
4	0,000021	0,02
6	0,0004315	0,022
8	0,0365767	0,0783
10	3,1185	6,2666

Tabla 3. Tiempos de ejecución Python toma 1.

Python Toma 2		
Ciudades	Tiempo (S) Secuencial	Tiempo(S) Paralelo
2	0,0000107	0,021
4	0,00002	0,02
6	0,0007591	0,023
8	0,0388274	0,0726
10	3,0672	5,9634

Tabla 4. Tiempos de ejecución Python toma 2.

VII. CONCLUSIONES

Con base en las gráficas proporcionadas, que representan el tiempo de ejecución de un algoritmo para resolver el problema del viajero (TSM) en Java y Python, se pueden sacar las siguientes conclusiones:

1. En ambas gráficas (Java y Python), el tiempo de ejecución incrementa de manera exponencial a medida que aumenta el número de ciudades, particularmente a partir de 8 ciudades. Esto es consistente con la naturaleza del problema, que tiene una complejidad factorial (O(n!)) para soluciones exactas.

2. Java:
   - Para un número menor de ciudades (≤ 8), tanto la ejecución secuencial como la paralela tienen tiempos muy similares, lo que sugiere que la sobrecarga del paralelismo no es significativa en problemas pequeños.
   - Sin embargo, para más de 8 ciudades, la ejecución paralela tiene un peor rendimiento que la secuencial. Esto podría deberse a la sobrecarga adicional del manejo de hilos o a una distribución ineficiente de las tareas.

3. Python:
   - Al igual que en Java, la ejecución secuencial y paralela tienen tiempos similares para casos pequeños.
   - Para más de 8 ciudades, la ejecución paralela también incrementa su tiempo de manera más rápida que la secuencial, aunque la diferencia en Python es más notoria en comparación con Java.
   - Para la versión secuencial, se evidencia un tiempo de ejecución mejor a la versión secuencial de Java.

4. Para el mismo número de ciudades, los tiempos en Python (especialmente para la ejecución paralela) son considerablemente mayores que los de Java. Esto podría deberse a:
   - La implementación subyacente del paralelismo (por ejemplo, el Global Interpreter Lock en Python).
   - Diferencias en el desempeño de las bibliotecas utilizadas.
   - El overhead adicional de Python como lenguaje interpretado, en contraste con Java, que es compilado.

5. En ambas implementaciones, el paralelismo no parece aportar una ventaja significativa en términos de tiempo para resolver el problema en instancias pequeñas y medianas. De hecho, resulta menos eficiente para casos grandes. Esto puede sugerir que:
   - La implementación paralela no está completamente optimizada.
   - La distribución de tareas no compensa el overhead asociado con la sincronización y comunicación entre hilos o procesos.

6. Para resolver este problema, Java parece ser más eficiente que Python en este caso específico, pero la elección final dependerá de otros factores como la facilidad de implementación para una versión más optimizada y que contemple una cantidad mayor de ciudades.

VIII. BIBLIOGRAFÍA

[1] “Algoritmo genético”. MathWorks - Creador de MATLAB y Simulink - MATLAB y Simulink - MATLAB & Simulink. Accedido el 16 de octubre de 2024. [En línea]. Disponible: https://es.mathworks.com/discovery/genetic-algorithm.html

[2] La universidad en internet. “La computación paralela: características, tipos y usos”. UNIR. Accedido el 16 de octubre de 2024. [En línea]. Disponible: https://www.unir.net/revista/ingenieria/computacion-paralela/

[3] ¿Qué es el Travelling Salesman Problem (TSP) y cómo solucionarlo? (s/f). Simpliroute. Recuperado el 1 de diciembre de 2024, de https://simpliroute.com/es/blog/que-es-el-travelling-salesman-problem-tsp-y-como-solucionarlo

[4] AlphaOpt [@alphaopt2024]. (s/f). What is the Traveling Salesman Problem? Youtube. Recuperado el 1 de diciembre de 2024, de https://www.youtube.com/watch?v=1pmBjIZ20pE

[5] Vargas, E. M., Martínez, M. R., Castillo, L. R. M., & Solis, L. S. (n.d.). *Implementación paralela de un algoritmo genético para el problema del agente viajero usando OpenMP*. Ipn.Mx. Retrieved December 2, 2024, from https://rcs.cic.ipn.mx/2016_128/Implementacion%20paralela%20de%20un%20algoritmo%20genetico%20para%20el%20problema%20del%20agente%20viajero.pdf