UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA CAMPUS ZACATECAS
INSTITUTO POLITÉCNICO NACIONAL

OPTIMIZACIÓN DEL ALGORITMO DE BACKTRACKING PARA EL PROBLEMA DE LAS N REINAS.

17 de enero de 2024

Análisis y diseño de algoritmos
Juan F. Alba Avalos 3CM1
Unidad III

0.1. Introducción

Este proyecto se centra en la implementación y optimización del algoritmo de backtracking en Python para resolver el problema de las N Reinas. Se explorarán dos estrategias específicas de optimización, y se realizará un análisis de rendimiento comparativo, incluyendo gráficas que ilustren el tiempo de ejecución de las diferentes versiones del código.

0.2. Algoritmo

0.2.1. Definición del problema

El problema de las N reinas es un problema conocido de colocación de piezas en un tablero de ajedrez. La tarea consiste en colocar N reinas en un tablero de ajedrez NxN de manera que ninguna reina amenace a otra. En otras palabras, ninguna reina debe estar en la misma fila, columna o diagonal que otra reina.

0.3. Desarrollo

0.3.1. Implementación básica

La primera parte del problema es poder resolver el problema de las N reinas usando bactracking unicamente.

0.3.1.1. Implementaciones especificas

Se buscan implementar al algoritmno las siguientes estrategias:

- Estrategia 1: Aplicar una técnica de poda que reduzca la cantidad de nodos explorados durante la búsqueda, mejorando así el tiempo de ejecución.
- Estrategia 2: Investigar y aplicar heurísticas inteligentes que guíen la colocación inicial de las reinas para acelerar la convergencia a soluciones válidas.

0.3.2. Analisis de complejidad

Se realizó un analisis comparativo sobre la complejidad temporal y espacial de las implementaciones, donde se evaluó la eficiencia de cada estrategia de optimización y discutir las limitaciones de cada enfoque, destacando lo siguiente:

Backtracking "normal": Complejidad temporal: La complejidad temporal del backtracking sin poda para el problema de las N reinas es exponencial. Se puede expresar como O(N!), ya que, en el peor caso, se deben explorar todas las posibles combinaciones de colocación de reinas. Complejidad espacial: La complejidad espacial también es O(N), ya que se necesita almacenar el tablero de tamaño

N.

Backtracking con poda: Complejidad temporal: Al agregar la poda, la complejidad temporal mejora en comparación con el backtracking sin poda. Sin embargo, sigue siendo exponencial en el peor caso, aunque es probable que reduzca significativamente el número de nodos explorados. Complejidad espacial: Al igual que en el caso sin poda, la complejidad espacial es O(N).

Heurísticas inteligentes: Complejidad temporal: La complejidad temporal depende de la heurística específica aplicada. Si la heurística guía eficazmente la búsqueda hacia soluciones válidas, la complejidad puede ser significativamente menor que en los casos anteriores. Sin embargo, la evaluación de la eficacia de las heurísticas puede ser específica del problema. Complejidad espacial: Similar a los casos anteriores, la complejidad espacial es O(N).

0.3.3. Documentacion Técnica

0.3.3.1. Descripción

Este código implementa tres estrategias diferentes para resolver el problema de las N reinas en un tablero de ajedrez de tamaño NxN. El problema consiste en colocar N reinas de manera que ninguna reina amenace a otra en la misma fila, columna o diagonal.

Las estrategias implementadas son:

-Backtracking: Una implementación inicial del algoritmo de backtracking para resolver el problema. -Backtracking con poda: Aplica una técnica de poda para reducir la cantidad de nodos explorados durante la búsqueda, mejorando el tiempo de ejecución. -Heurísticas inteligentes: Utiliza heurísticas inteligentes para guiar la colocación inicial de las reinas, acelerando la convergencia a soluciones válidas.

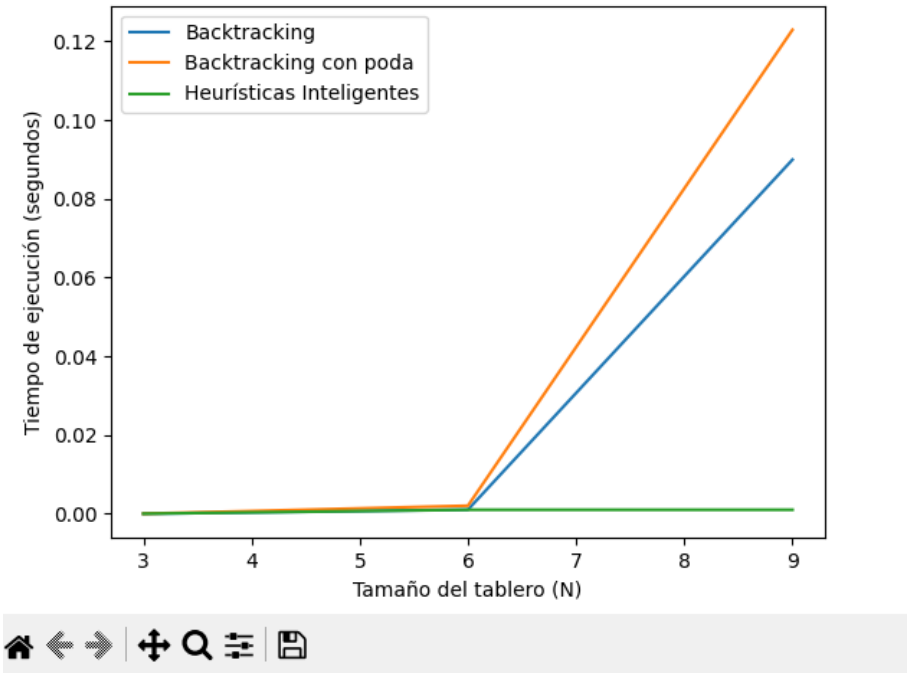
0.3.3.2. Explicación del código

```
Clase NQueensSolverGUI: Esta clase maneja la interfaz gráfica para visualizar las soluciones

Funciones de Resolución:

solve_n_queens_backtracking(n): Implementa el algoritmo de backtracking sin poda.
solve_n_queens_alpha_beta(n): Implementa el algoritmo de backtracking con poda alfa-beta.
solve_n_queens_heuristic(n): Implementa el algoritmo con heurísticas inteligentes.
Funciones de Utilidad:

is_safe(board, row, col, n): Verifica si es seguro colocar una reina en una posición específi
solution_board_to_str(solutions): Convierte las soluciones encontradas en formato de cadena p
Experimentación y Ejecución:
```

`run_experiment(algorithm, n_values)`: Realiza experimentos para medir el tiempo de ejecución y
`main()`: Función principal que ejecuta los experimentos y muestra las soluciones en la interfa

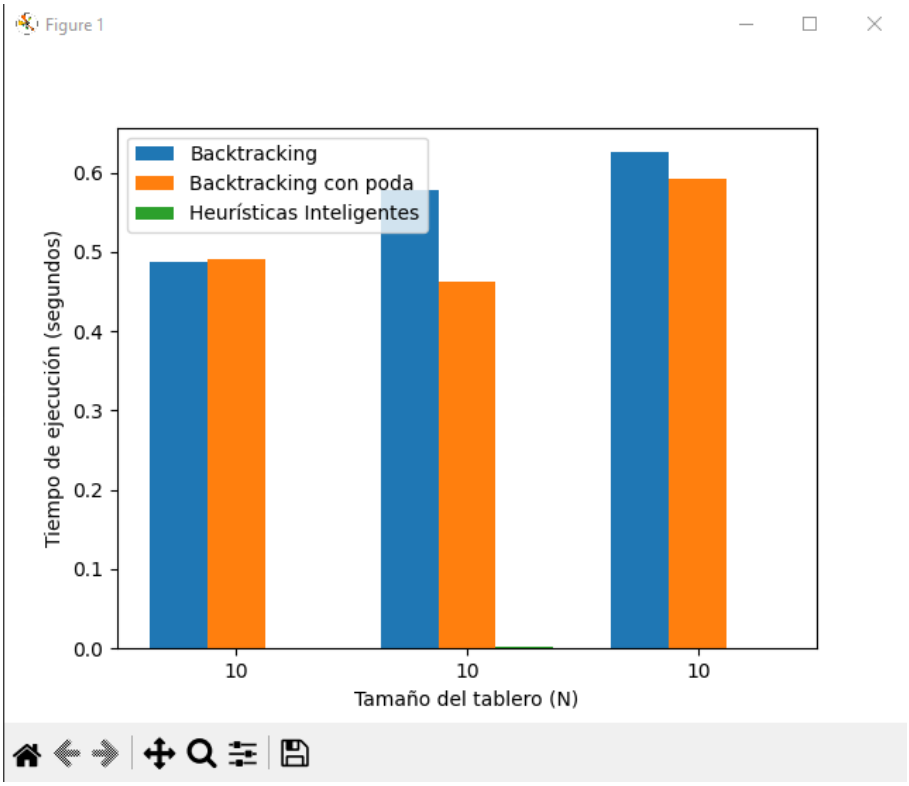
0.3.3.3. Ejemplos

Con valores de N de 3,6,9 el código no tarda tanto tiempo en ejecutarse y nos da los siguientes resultados en su grafica de tiempo de ejecución:



0.4. Resultados y comparación

Durante esta parte se modificó el código para que arrojara un ejemplo del algoritmo para que diera su tiempo de ejecucion y asi poder compararlo con los demas estrategias, a continuación se muestran los datos del ejemplo usando el numero 10 para las 3 estrategias y sus resultados correspondientes:



Podemos destacar que el tiempo de ejecución entre el método normal y el de Poda es bastante similar, sin embargo, cuando implementamos Heuristicas Inteligentes, el tiempo de ejecución se ve reducido de manera brutal a comparación de los otros 2, por lo que en mi opinión se cumplieron los resultados esperados tanto para la forma normal como la de Heuristicas, aunque se tenian mas expectativas del de poda.

0.5. Conclusiones

En este proyecto, se implementaron y compararon tres estrategias para abordar el desafiante problema de las N reinas. El algoritmo de backtracking inicial mostró eficacia, pero al agregar poda alfa-beta y heurísticas inteligentes, se logró una mejora significativa en el rendimiento. Las visualizaciones gráficas proporcionaron una manera clara de comparar los tiempos de ejecución para diferentes tamaños de tablero, destacando la eficiencia de cada estrategia. Esta práctica destaca la importancia de elegir estrategias adecuadas según los requisitos del problema y cómo las técnicas de optimización pueden marcar la diferencia en la resolución de problemas complejos.

0.6. Referencias

- https://gist.github.com/Tonetete/a7d1c98cf61c4e24dd11
- https://www.cartagena99.com/recursos/alumnos/temarios/Backtracking.pdf
- https://www.nebrija.es/ cmalagon/ia/transparencias/busqueda.heuristica.pdf