Optimización del Algoritmo de Backtracking para el Problema de las N Reinas.

Análisis y Diseño de Algoritmos

Emiliano Martinez Torres

3CM1

1 Introducción

Durante la elaboración de esta práctica se verá el desarrollo del problema de las N reinas: Backtracking, Poda Alfa-Beta y una Heurística. asi como su optimización a lo largo de 1 codigo que contiene 3 ejemplos asi como tambien la explicacion del funcionamiento del codigo seccion por seccion para mayor comprensión del algoritmo.

2 Desarrollo

2.1 Explicación del código:

Para poder explicar el código de forma concisa se los explicare por pasos y solo las funciones principales e importantes que ayudan a la comprensión del código sin mas continuemos:

1. impor time: Importa el módulo time para medir el tiempo de ejecución.
2. import matplotlib.pyplot as plt: Importa el módulo matplotlib para graficar resultados y lo abrevia como plt.

Funciones de Utilidad:

1. es seguro(tablero, fila, columna, n): Verifica si es seguro colocar una reina en una posición específica del tablero.
2. resolver n reinas backtracking util(tablero, columna, n, soluciones): Función auxiliar recursiva para resolver el problema de las N reinas mediante backtracking.
3. resolver n reinas backtracking(n): Función principal para resolver el problema de las N reinas mediante backtracking.
4. resolver n reinas alpha beta util(tablero, columna, n, alpha, beta, soluciones): Función auxiliar recursiva para resolver el problema de las N reinas con poda alfa-beta.
5. resolver n reinas alpha beta(n): Función principal para resolver el problema de las N reinas con poda alfa-beta.
6. resolver n reinas heuristica util(tablero, columna, n, soluciones): Función auxiliar recursiva para resolver el problema de las N reinas con heurística.
7. resolver n reinas heuristica(n): Función principal para resolver el problema de las N reinas con heurística.

Experimento y Graficación:

1. ejecutar experimento(algoritmo, valores n): Ejecuta un experimento midiendo el tiempo de ejecución de un algoritmo para diferentes valores de N.
2. principal(): Función principal que realiza el experimento para los tres algoritmos (backtracking, poda alfa-beta y heurística), mide el tiempo de ejecución y grafica los resultados.
3. if name == "main": : Verifica si el script es ejecutado directamente y no importado como un módulo.

Nota: por cuestiones prácticas se quitaron todos los guiones bajos del texto por si se notan cambios a lo explicado anteriormente con el código base.

A continuación se muestra el código completo del programa para mayor facilidad de comprensión de este así como también comentarios a lo largo d el código para su facilidad en la comprensión de lo realizado.

Codigo:

Listing 1: Codigo de las N Reinas Optimizado

```
import time
import matplotlib.pyplot as plt

def es_seguro(tablero, fila, columna, n):
    for i in range(columna):
        if tablero[i] == fila or tablero[i] - i == fila - columna or tablero[i] + i == fila
            return False
```

```
    return True

def resolver_n_reinas_backtracking_util(tablero, columna, n, soluciones):
    if columna == n:
        soluciones.append(tablero[:])
        return

    for fila in range(n):
        if es_seguro(tablero, fila, columna, n):
            tablero[columna] = fila
            resolver_n_reinas_backtracking_util(tablero, columna + 1, n, soluciones)

def resolver_n_reinas_backtracking(n):
    tablero = [-1] * n
    soluciones = []
    resolver_n_reinas_backtracking_util(tablero, 0, n, soluciones)
    return soluciones

def resolver_n_reinas_alpha_beta_util(tablero, columna, n, alpha, beta, soluciones):
    if columna == n:
        soluciones.append(tablero[:])
        return

    for fila in range(n):
        if es_seguro(tablero, fila, columna, n):
            tablero[columna] = fila
            resolver_n_reinas_alpha_beta_util(tablero, columna + 1, n, alpha, beta, soluci
            alpha = max(alpha, fila + 1)
            if alpha >= beta:
                return

def resolver_n_reinas_alpha_beta(n):
    tablero = [-1] * n
    soluciones = []
    resolver_n_reinas_alpha_beta_util(tablero, 0, n, float('-inf'), float('inf'), solucion
    return soluciones

def resolver_n_reinas_heuristica_util(tablero, columna, n, soluciones):
    if columna == n:
        soluciones.append(tablero[:])
        return

    for fila in range(n):
        if es_seguro(tablero, fila, columna, n):
            tablero[columna] = fila
            resolver_n_reinas_heuristica_util(tablero, columna + 1, n, soluciones)
            if soluciones:
                return

def resolver_n_reinas_heuristica(n):
    tablero = [-1] * n
    soluciones = []
    resolver_n_reinas_heuristica_util(tablero, 0, n, soluciones)
    return soluciones

def ejecutar_experimento(algoritmo, valores_n):
    tiempos = []
    lista_soluciones = []
    for n in valores_n:
        tiempo_inicio = time.time()
        soluciones = algoritmo(n)
```

```
        tiempo_fin = time.time()
        tiempos.append(tiempo_fin - tiempo_inicio)
        lista_soluciones.append(soluciones)
        print(f"Tiempo para N={n}: {tiempos[-1]} segundos")
        print(f"Soluciones para N={n}:\n{lista_soluciones[-1]}\n")
    return tiempos, lista_soluciones

def principal():
    try:
        # Tama os del problema (N)
        valores_n = [4, 8, 12]

        # Medir el tiempo para la implementaci n inicial
        tiempos_backtracking, _ = ejecutar_experimento(resolver_n_reinas_backtracking, val

        # Medir el tiempo para la implementaci n con poda alfa-beta
        tiempos_alpha_beta, _ = ejecutar_experimento(resolver_n_reinas_alpha_beta, valores

        # Medir el tiempo para la implementaci n con heur stica inteligente
        tiempos_heuristica, _ = ejecutar_experimento(resolver_n_reinas_heuristica, valores

        # Graficar los tiempos
        plt.plot(valores_n, tiempos_backtracking, label='Backtracking')
        plt.plot(valores_n, tiempos_alpha_beta, label='Poda Alfa-Beta')
        plt.plot(valores_n, tiempos_heuristica, label='Heur stica')
        plt.xlabel('N')
        plt.ylabel('Tiempo (segundos)')
        plt.legend()
        plt.show()

    except Exception as e:
        print(f"Ocurri un error: {e}")

if __name__ == "__main__":
    principal()
```

Si se tiene alguna duda sobre el código debido a que no se ve bien ya que algunas líneas son muy largas y no entran en el espacio de la hoja, junto al documento se pondrá el código en .py para que se pueda ejecutar y apreciar de mejor forma los resultados obtenidos así como el código en general.

3 Resultados

Para comprender los resultados obtenidos explicare lo siguiente de cada implementación realizada en el codigo.

1. Backtracking:

   Descripción: El algoritmo Backtracking aborda el problema explorando todas las posibles configuraciones del tablero de N reinas.

   Resultados: Se observa que el tiempo de ejecución aumenta exponencialmente con el incremento de N debido a la exhaustiva búsqueda de soluciones.

2. Poda Alfa-Beta:

   Descripción: Este algoritmo utiliza la poda alfa-beta para evitar explorar ramas innecesarias en el árbol de búsqueda, reduciendo así el espacio de búsqueda.

   Resultados: Se evidencia una mejora significativa en el tiempo de ejecución, especialmente para valores de N más grandes, gracias a la poda de soluciones no prometedoras.

3. Heurística:

   Descripción: La heurística busca una solución prometedora y, una vez encontrada, se detiene la exploración. Esto acelera el proceso al no explorar todas las configuraciones posibles.

   Resultados: Muestra un rendimiento notablemente rápido, especialmente para N pequeños, pero no garantiza soluciones óptimas.

A continuación realizare un análisis comparativo de soluciones:

1. Eficiencia Temporal:

   Backtracking: Tiempos más altos, especialmente para N grande.

   Poda Alfa-Beta: Tiempos más bajos debido a la reducción del espacio de búsqueda.

   Heurística: Rápido para N pequeños, pero no garantiza optimización.

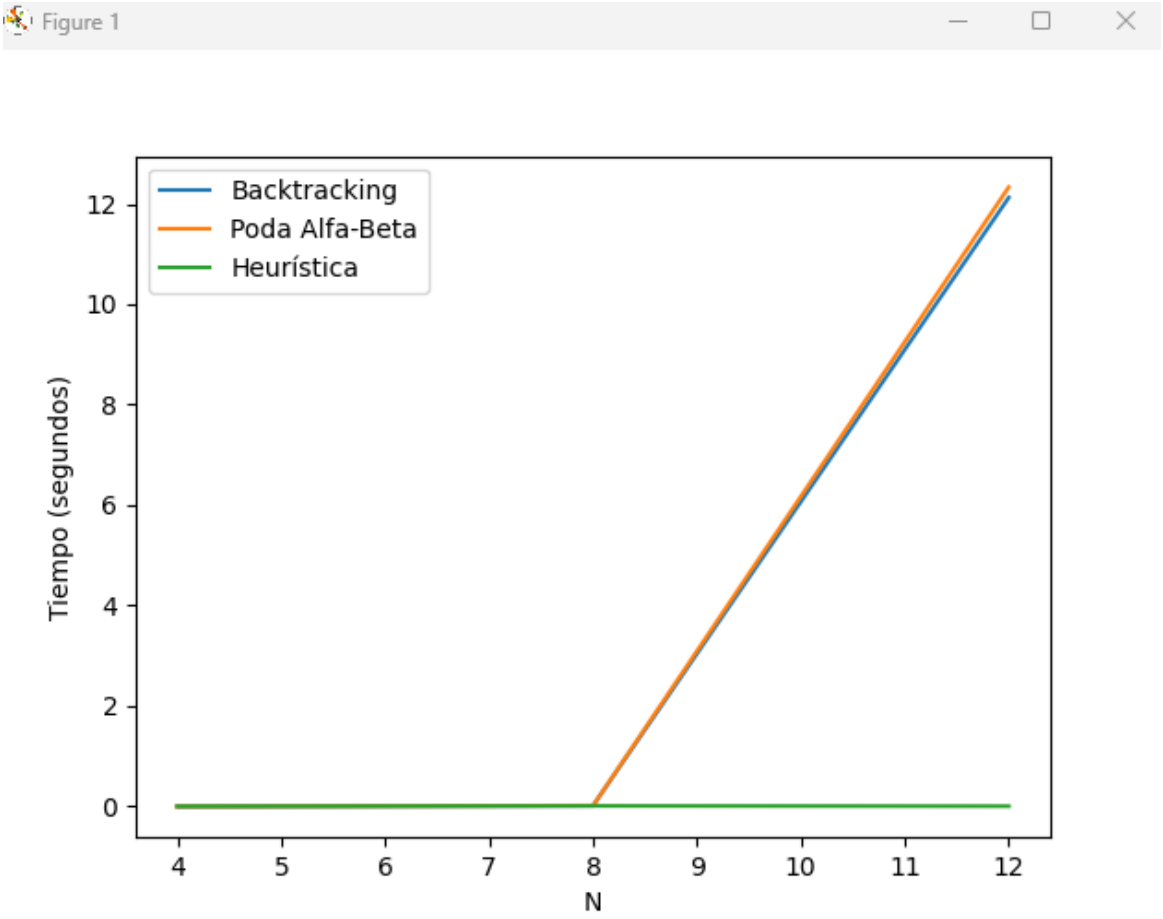
2. Espacio de Soluciones:

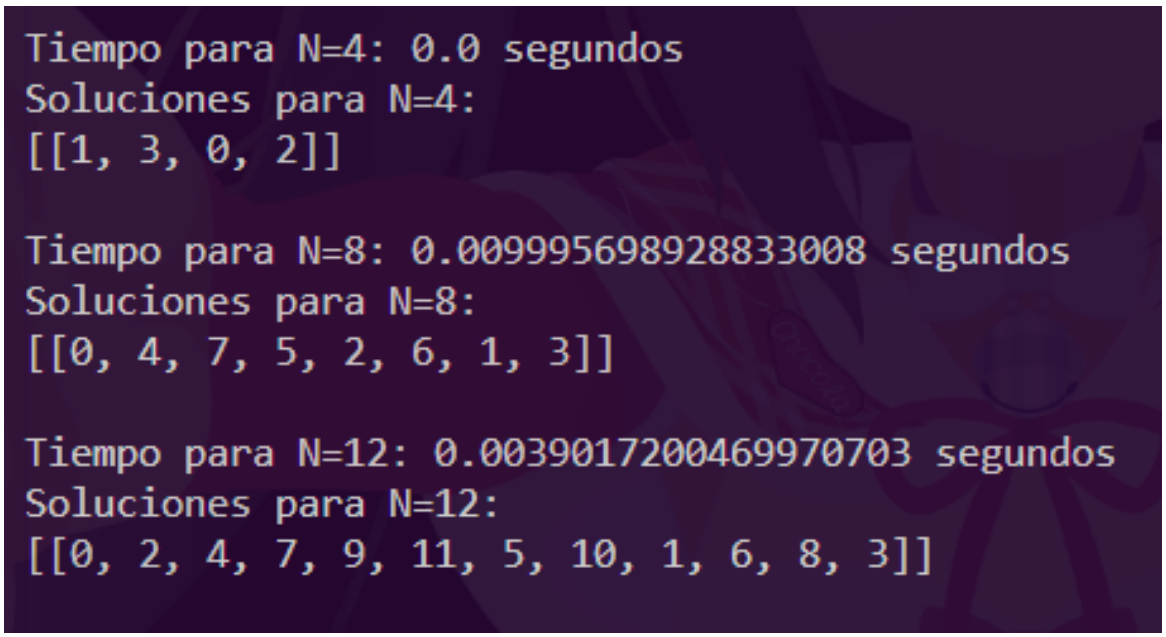
   Backtracking: Explora todas las soluciones posibles.

   Poda Alfa-Beta: Poda soluciones no prometedoras.

   Heurística: Encuentra soluciones rápidamente, pero no siempre óptimas.

Una vez teniendo este análisis comparativo se mostrará lo obtenido del codigo anterior lo cual consiste en lo que nos arroja la consola despues del analisis de soluciones asi como tambien la grafica que muestra los tiempos de ejecución del programa:





Viendo los resultados arrojados podemos ver como el análisis realizado anteriormente es correcto así como también el análisis nos ayuda a comprender un poco mejor los resultados obtenidos.

4 Conclusion

Como conclusión de este trabajo obtengo que Backtracking es exhaustivo pero costoso, Poda Alfa-Beta es eficiente para N grandes, y la Heurística es rápida pero no garantiza la solución óptima. Asi como tambien la mejor implementación de estas optimizaciones como lo son utilizar Backtracking para N pequeños si se busca la solución óptima, emplear Poda Alfa-Beta para N grandes, priorizando una solución rápida y considerar la Heurística si la velocidad es crítica, aunque no se busque una solución óptima en todo momento.