Instituto Politécnico Nacional

UPIIZ - Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas

Proyecto Final

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

Giovanna Inosuli Campos Flores

17 de enero de 2024

1 Introducción

El problema de las N reinas es un famoso desafío en el campo de la teoría de juegos y la inteligencia artificial, especialmente en la resolución de problemas de búsqueda y optimización. En este problema, se busca colocar N reinas en un tablero de ajedrez de dimensiones N x N, de manera que ninguna reina amenace a otra. La formulación del problema se presenta a continuación:

- Variables: $\{x_i\}, i = 1..N$.
- Dominio: $\{1, 2, 3, ..., N\}$ para todas las variables.
- Restricciones:
  - $\forall x_i, x_j, i \neq j : x_i \neq x_j$ (No en la misma fila).
  - $\forall x_i, x_j, i \neq j : x_i - x_j \neq i - j$ (No en la misma diagonal SE).
  - $\forall x_i, x_j, i \neq j : x_j - x_i \neq i - j$ (No en la misma diagonal SO).

La dificultad de este problema radica en encontrar configuraciones que cumplan con estas restricciones. Diversos algoritmos, como búsqueda exhaustiva, algoritmos genéticos, búsqueda heurística y algoritmos de retroceso, se han utilizado para abordar este problema. El algoritmo de retroceso es uno de los métodos más comunes para resolver el problema de las N reinas. El backtracking consiste en explorar en profundidad el espacio de búsqueda, instanciando sucesivamente las variables y verificando la consistencia local. Si se encuentra una inconsistencia, el algoritmo retrocede y prueba una configuración diferente. Este proceso continúa hasta encontrar una solución válida o explorar todas las posibilidades.

2 Desarrollo

2.1 Implementación Básica

El objetivo es encontrar todas las disposiciones posibles de N reinas en un tablero de ajedrez N×N de manera que ninguna de las reinas amenace a las demás. se utilizó un enfoque recursivo de backtracking para explorar diferentes soluciones.

```python
# Función recursiva para resolver el problema de las N reinas
def backtrack(fila_actual, n, contador):
    global columna
    global diagonal_izquierda
    global diagonal_derecha

    if fila_actual== n:
        #retorna
        return contador + 1

    for x in range(n):
        if columna[x] or diagonal_izquierda[x + fila_actual] or diagonal_derecha[x - fila_actual + n - 1]:
            continue
        #colocamos una reina
        columna[x] = diagonal_izquierda[x + fila_actual] = diagonal_derecha[x - fila_actual + n - 1] = True
        #enviamos la fila siguiente
        contador = backtrack(fila_actual + 1, n, contador)
        #quitamos la reina para probar otras posibilidades
        columna[x] = diagonal_izquierda[x + fila_actual] = diagonal_derecha[x - fila_actual + n - 1] = False

    return contador
```

Figura 1: Ejemplo de Funcion

Se inicializan variables: n es el tamaño del tablero de ajedrez, *contador* es un contador para realizar un seguimiento del número de soluciones encontradas, y se utilizan tres arreglos (`columna`, `diagonal_izquierda`, `diagonal_derecha`) para verificar si se puede colocar una reina en una posición particular. La función `backtrack` toma tres parámetros: la fila actual, n y *contador*. Luego se verifica el caso base: Si la fila actual alcanza n, significa que todas las reinas se han colocado con éxito y se incrementa *contador* en 1. Luego, la función retorna el *contador* actualizado.

Para cada columna en la fila actual (x), se verifica si colocar una reina en esa posición es válida según las condiciones en la declaración `if`. Si es válida, se coloca la reina y se llama a la función recursivamente para la siguiente fila (`fila_actual+1`).

Después de la llamada recursiva, se quita la reina (backtracking) para explorar otras posibilidades. El recuento final de soluciones se imprime llamando a `backtrack`.

Este fragmento de código forma parte de la lógica principal del algoritmo de backtracking para resolver el problema de las N reinas. Este algoritmo explora de manera recursiva todas las posibles combinaciones de colocar reinas en el tablero, y el backtracking se utiliza para deshacer decisiones anteriores y probar otras opciones cuando se llega a una solución parcial.

2.2 Optimizaciones Específicas

2.2.1 Estrategia

La *poda por factibilidad* es una técnica utilizada en algoritmos de búsqueda para reducir la cantidad de nodos explorados, evitando la exploración de ramas que sabemos que no conducirán a una solución viable. En el contexto del problema de las N reinas, la poda por factibilidad implica evitar la exploración de ciertas configuraciones del tablero que no cumplen con las restricciones del problema.

En el algoritmo de backtracking para las N reinas, la poda por factibilidad se implementa mediante la verificación de las condiciones antes de explorar una rama específica. Por ejemplo, al colocar una reina en una posición particular, se puede verificar de inmediato si esta decisión viola alguna restricción, como la condición de no estar en la misma fila, columna o diagonal que otra reina.

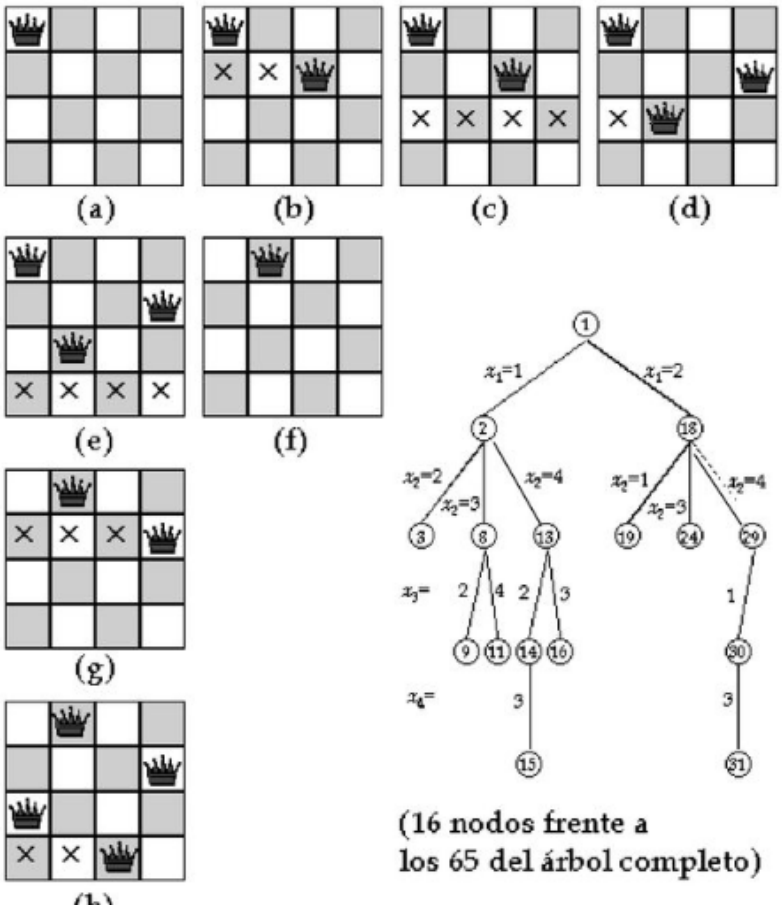

Figura 2: Ejemplo

Si se detecta que la configuración actual no cumple con las restricciones, se evita la exploración adicional de esa rama específica, lo que contribuye a mejorar la eficiencia del algoritmo al reducir la cantidad de configuraciones inválidas consideradas.

La poda por factibilidad es esencial para abordar problemas combinatorios complejos, como el problema de las N reinas, donde la exploración exhaustiva de todas las posibles combinaciones puede ser computacionalmente costosa.

2.2.2 Estrategia

Una heurística inteligente es una técnica que ayuda a encontrar una solución aproximada a un problema, pero no garantiza que sea la solución óptima. El algoritmo llamado Las Vegas es inteligente porque utiliza una estrategia de generación aleatoria para encontrar soluciones al problema de las N reinas.

Se explica que el backtracking implica probar todos los caminos posibles para encontrar una solución, mientras que Las Vegas utiliza decisiones aleatorias para apostar por la complejidad o tiempo de ejecución del algoritmo. Se menciona que Las Vegas puede ser eficiente a pesar de su aleatoriedad, y se presenta un ejemplo detallado de cómo el algoritmo coloca las reinas de manera aleatoria en el tablero, teniendo en cuenta las restricciones para evitar ataques.

Las heurísticas inteligentes se basan en la idea de que, si se genera una gran cantidad de soluciones aleatorias, es probable que se encuentre una solución válida. El algoritmo Las Vegas genera soluciones aleatorias hasta que encuentra una solución válida.

```python
def disponibles(y, n, columna, diagonal_izquierda, diagonal_derecha):
    # Devuelve las columnas no atacadas en la fila y
    disponibles = [x for x in range(n) if not columna[x] and not diagonal_izquierda[x + y]
                   and not diagonal_derecha[x - y + n - 1]]
    return disponibles

def generar_tablero_aleatorio(n):
    # Genera una disposición aleatoria de reinas en un tablero de tamaño n x n
    solucion = []
    columna = [False] * n
    diagonal_izquierda = [False] * (n * 2)
    diagonal_derecha = [False] * (n * 2)

    for y in range(n):
        columnas_disponibles = disponibles(y, n, columna, diagonal_izquierda, diagonal_derecha)
        if columnas_disponibles:
            x = random.choice(columnas_disponibles)
        else:
            break
        columna[x] = diagonal_izquierda[x + y] = diagonal_derecha[x - y + n - 1] = True
        solucion.append((x, y))
```

Figura 3: Algoritmo de las Vegas

El código del algoritmo Las Vegas se presenta, destacando la función para determinar las columnas disponibles en cada fila y la elección aleatoria de las mismas. Se explica cómo el algoritmo se ejecuta hasta que se coloca una reina en cada fila, reiniciando el tablero en caso de que no haya columnas disponibles en una fila.

3 Resultados

Se proporciona una comparación de los tiempos de ejecución entre Las Vegas y el método de backtracking. Se muestra que Las Vegas tiene tiempos de ejecución más bajos, especialmente en tableros más grandes, y se destaca su eficiencia a pesar de la aleatoriedad. se observo que en Backtrack se puede concluir que, entre tableros más pequeños, como el de tamaño 6, pueden resolverse en tiempos muy cortos y con un número manejable de soluciones. A medida que el tamaño del tablero aumenta, el tiempo de ejecución y el número de soluciones aumentan rápidamente

```
PS C:\Users\giova\OneDrive\Documentos\Analisis> & C:/Users/giova/AppData/Local/Micros
isis/Proyecto_final/4/1.py
Tamaño del tablero: 11, Soluciones Encontradas: 2680, Tiempo: 0.590909 segundos
Tamaño del tablero: 10, Soluciones Encontradas: 724, Tiempo: 0.148336 segundos
Tamaño del tablero: 12, Soluciones Encontradas: 14200, Tiempo: 2.744523 segundos
Tamaño del tablero: 15, Soluciones Encontradas: 2279184, Tiempo: 580.740885 segundos
Tamaño del tablero: 6, Soluciones Encontradas: 4, Tiempo: 0.001004 segundos
```

Figura 4: Prueba de Backtrack

El número de soluciones encontradas aumenta con según el tamaño del tablero, lo cual es esperado. Cada vez hay más formas de colocar las reinas en tableros más grandes. La diferencia entre el número de soluciones para tamaños de tablero adyacentes puede variar, ya que algunos tamaños pueden ser más propicios para ciertos arreglos. En el tiempo de ejecución aumenta significativamente a medida que el tamaño del tablero (N) aumenta. Especialmente notorio es el tiempo requerido para un tablero de tamaño 15, que es considerablemente mayor debido a la explosión combinatoria de posibles disposiciones. Se puede decir que entre mayor sea el tablero mayor será el número de soluciones y consecuencia el tiempo aumenta

En caso del algoritmo Las Vegas es más eficiente que el algoritmo de backtracking, ya que no tiene que explorar todas las posibles configuraciones de reinas. Sin embargo, el

algoritmo Las Vegas puede tardar más tiempo en encontrar una solución, ya que depende de la suerte. En este caso se observó que la vegas pude resolver tableros de mayor tamaño en menor tiempo a comparaciones de backtrack especialmente para tableros superiores a 100 se observa que su tiempo de ejecución es bajo.

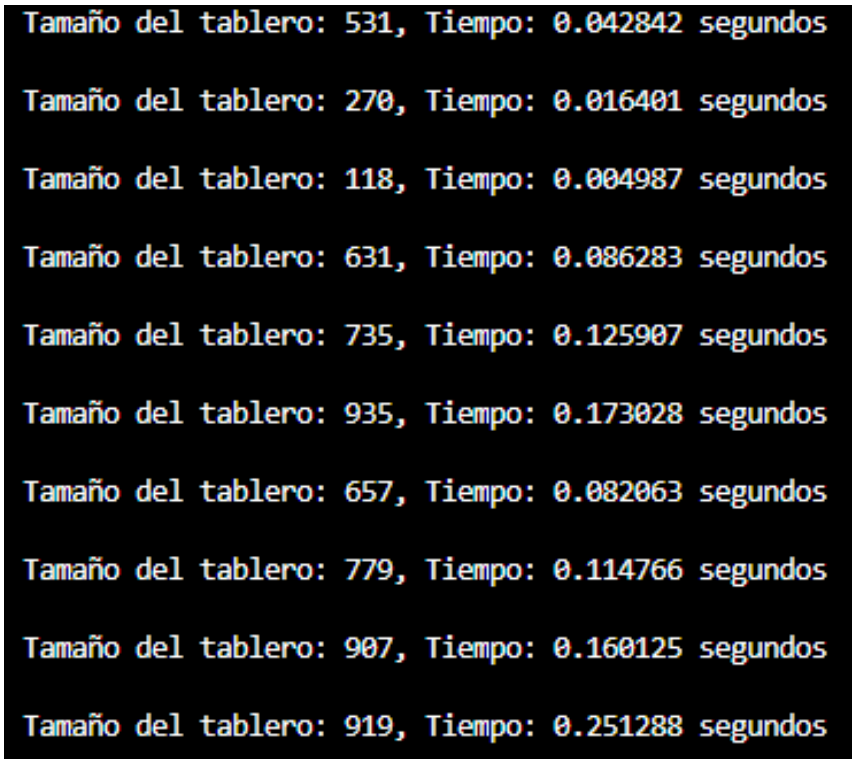

Figura 5: Soluciones del Algoritmo de las Vegas

Por lo que se concluye que principal diferencia entre el algoritmo de Las Vegas y el backtracking es que el algoritmo de Las Vegas es un algoritmo probabilístico, mientras que el backtracking es un algoritmo determinista. El algoritmo de backtracking funciona explorando todas las posibles configuraciones de reinas, hasta que encuentra una solución válida. El algoritmo de Las Vegas, por otro lado, genera una configuración aleatoria de reinas y luego verifica si es válida. Si la configuración no es válida, el algoritmo de Las Vegas genera otra configuración aleatoria. El algoritmo de Las Vegas continúa generando configuraciones aleatorias hasta que encuentra una configuración válida.

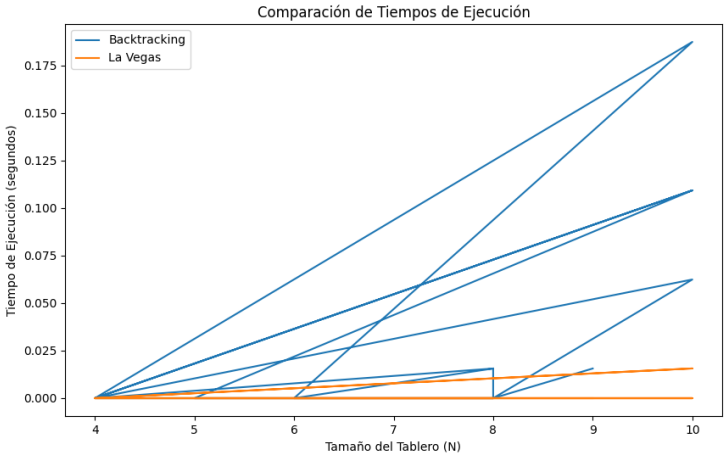

Figura 6: Comparación de tiempos

4 Conclusiones

La complejidad computacional del problema aumenta exponencialmente con el tamaño del tablero, haciéndolo interesante para el estudio y desarrollo de algoritmos eficientes. Además, la resolución del problema tiene aplicaciones en sistemas de ajedrez, diseño de circuitos y otras situaciones donde es necesario colocar objetos sin amenazarse entre sí. el algoritmo de backtracking utilizado para resolver el problema de las N reinas, la complejidad de tiempo y espacio es exponencial en el peor caso. En casos de complejidad computacional, el algoritmo de Las Vegas es mejor que el backtracking para tableros pequeños. Esto se debe a que el algoritmo de Las Vegas no tiene que explorar todas las posibles configuraciones de reinas, lo que lo hace más eficiente. La complejidad computacional del algoritmo de backtracking es exponencial en el tamaño del tablero. Esto significa que el tiempo de ejecución del algoritmo de backtracking aumenta exponencialmente a medida que aumenta el tamaño del tablero. La complejidad computacional del algoritmo de Las Vegas es lineal en el tamaño del tablero. Esto significa que el tiempo de ejecución del algoritmo de Las Vegas aumenta linealmente a medida que aumenta el tamaño del tablero.

5 Bibliografía

M. D'Addario, Inteligencia Artificial: Tratados, Aplicaciones, Usos y Futuro. Independently Publ., 2019.

J. T. PALMA MÉNDEZ, Org., nteligencia Artificial: Métodos, técnicas y aplicaciones. ESPAÑA: Univ. Murcia.

VAN BEEK, Peter. Backtracking search algorithms. En Foundations of artificial intelligence. Elsevier, 2006. p. 85-134.