Unidad Profesional Interdisciplinaria de Ingeniería Campus Zacatecas Instituto Politécnico Nacional

UNIDAD 3

REPORTE

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

Nombre	Boleta
Alejandro Ulloa Reyes	2022670235
Enid Aimee Perez Robles	2023670020
Itzel Citlalli Flores Diaz	2023670093

Grupo:
3CM2

Docente:
M. en C. Erika Sánchez-Femat

1 Introducción

En el ámbito de la inteligencia artificial y la resolución de problemas computacionales, el problema de las N reinas ha sido un desafío clásico. Este problema consiste en situar N reinas en un tablero de ajedrez de tal manera que ninguna reina amenace a otra, es decir, ninguna reina comparta fila, columna o diagonal con otra. Una de las técnicas más eficientes para abordar este problema es el algoritmo de backtracking.

Este proyecto se centra en implementar y optimizar el algoritmo de backtracking utilizando Python como lenguaje de programación para resolver el desafío de las reinas. No solo nos limitaremos a la implementación básica..

A lo largo del desarrollo del proyecto, realizaremos un análisis comparativo del rendimiento entre las diferentes versiones del código. Este análisis estará respaldado por gráficos que mostrarán los tiempos de ejecución de cada variante, brindando una visión clara sobre la eficacia de las estrategias implementadas.

2 Desarrollo

Para empezar con el desarrollo de este reporte tenemos que definir los siguientes conceptos:

Algoritmo de Backtracking

El algoritmo de backtracking es una técnica poderosa para resolver problemas combinatorios y de búsqueda exhaustiva. Se utiliza para explorar todas las posibles soluciones de un problema de manera sistemática, descartando aquellas que no cumplen ciertas condiciones.

El backtracking es un enfoque recursivo que explora todas las opciones posibles para encontrar una solución a un problema. Funciona dividiendo el problema en subproblemas más pequeños, realizando pruebas y retrocediendo (backtracking) cuando se encuentra una solución inválida. Es especialmente útil para problemas donde es necesario probar muchas combinaciones diferentes para encontrar una solución óptima.

Problema de las N Reinas

El problema de las N Reinas consiste en colocar n reinas en un tablero de ajedrez de n x n de tal manera que no sea posible que dos reinas se capturen entre si, es decir, que no estén en la misma fila, ni en la misma columna ni en la misma diagonal. Se dice que hay una colisión si hay dos reinas que se pueden capturar entre si.

Se trata pues de encontrar una configuración de elegir las n celdas donde colocar a las reinas que minimice el número total de colisiones.

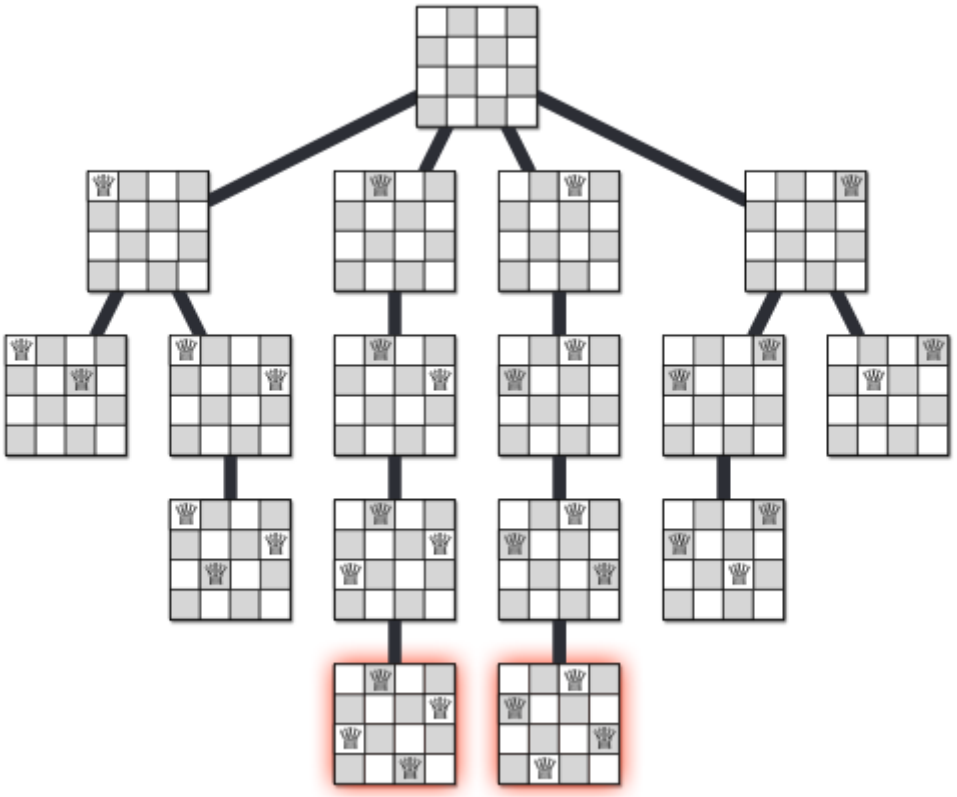

Figure 1: Ejemplo del problema de N Reinas

Hay soluciones analíticas que implican dar una fórmula explícita para la ubicación de las reinas o concatenar soluciones para valores menores de n. El problema con esta forma es que solo genera una cantidad muy pequeña de soluciones.

Aqui lo que se hizo fue crear un codigo aplicando el algoritmo de-backtracking que solucionara este problema más rapido y eficaz pero que al mismo timepo nos diera la solucion de un valor mas grande con un margen de soluciones mas amplio y no fuera tan tedioso.

Y ya el codigo final de este proyecto es el sigiente:

import time

```
import random
import matplotlib.pyplot as plt
def is_safe(board, row, col, n) :
for i in range(row) :
if board[i][col] == 1 :
return False

    for j in range(n):
if board[i][j] == 1 and abs(i - row) == abs(j - col):
return False

    return True

    def solve_nqueens_until(board, row, n) :
if row == n :
return [list(row) for row in board]

    solutions = []

    for col in range(n):
if is_safe(board, row, col, n) :
board[row][col] = 1
result = solve_nqueens_until(board, row + 1, n)

    if result:
solutions.extend(result)

    board[row][col] = 0

    return solutions
```

def solve$_{nq}ueens(n)$:
$board = [[0] * nfor_inrange(n)]$
$solve_{nq}ueens_util(board, 0, n)$

def calculate$_attacks(board, row, col, n)$:
$attacks = 0$

for i, j in zip(range(row - 1, -1, -1), range(col - 1, -1, -1)):
if board[i][j] == 1:
attacks += 1

for i, j in zip(range(row + 1, n), range(col - 1, -1, -1)):
if board[i][j] == 1:
attacks += 1

for i in range(n):
if i != row and board[i][col] == 1:
attacks += 1

for i, j in zip(range(row - 1, -1, -1), range(col + 1, n)):
if board[i][j] == 1:
attacks += 1

for i, j in zip(range(row + 1, n), range(col + 1, n)):
if board[i][j] == 1:
attacks += 1

return attacks

def hill$_c limbing_{nq} ueens(n, max_i terations = 1000)$:
$board = [[0] * nfor_i nrange(n)]$
$foriiinrange(n)$:
$j = random.randint(0, n - 1)$
$board[i][j] = 1$

current$_a ttacks = sum(calculate_a ttacks(board, row, col, n)forrow, coli$

for iteration in range(max$_i terations$) :
$ifcurrent_a ttacks == 0$:
$break$

min$_a ttacks = float('inf')$
$move = None$

for row in range(n):
for col in range(n):
if board[row][col] == 1:
continue

board[row][col] = 1
new$_a ttacks = sum(calculate_a ttacks(board, r, c, n)forr, cinenumerate(ra$
$board[row][col] = 0$

if new$_a ttacks < min_a ttacks$:
$min_a ttacks = new_a ttacks$
$move = (row, col)$
$ifmoveisNone$:
$returnboard$
$board[move[0]][move[1]] = 1$

$current_attacks = min_attacks$

def compare$_r untimes(max_q ueens = 15)$:
$backtracking_t imes = []$
$hill_c limbing_t imes = []$

for n in range(1, max$_q ueens + 1)$:
$Mediciónde tiempo para backtracking$
$start_t ime_b acktracking = time.time()$
$solve_{nq} ueens(n)$
$end_t ime_b acktracking = time.time()$
$backtracking_t imes.append(end_t ime_b acktracking - start_t ime_b acktrackin$

Medición de tiempo para búsqueda local
start$_t ime_h ill_c limbing = time.time()$
$hill_c limbing_{nq} ueens(n)$
$end_t ime_h ill_c limbing = time.time()$
$hill_c limbing_t imes.append(end_t ime_h ill_c limbing - start_t ime_h ill_c limbing)$

Graficar los resultados
plt.plot(range(1, max$_q ueens$+1), $backtracking_t imes, label =' Backtrackin$
$plt.plot(range(1, max_q ueens+1), hill_c limbing_t imes, label =' BúsquedaL$
$plt.xlabel('Número de reinas')$
$plt.ylabel('Tiempo de ejecución (segundos)')$
$plt.title('Comparación de tiempos de ejecución')$
$plt.legend()$
$plt.show()$

Llamada a la función para graficar
compare$_r untimes(max_q ueens = 10)$

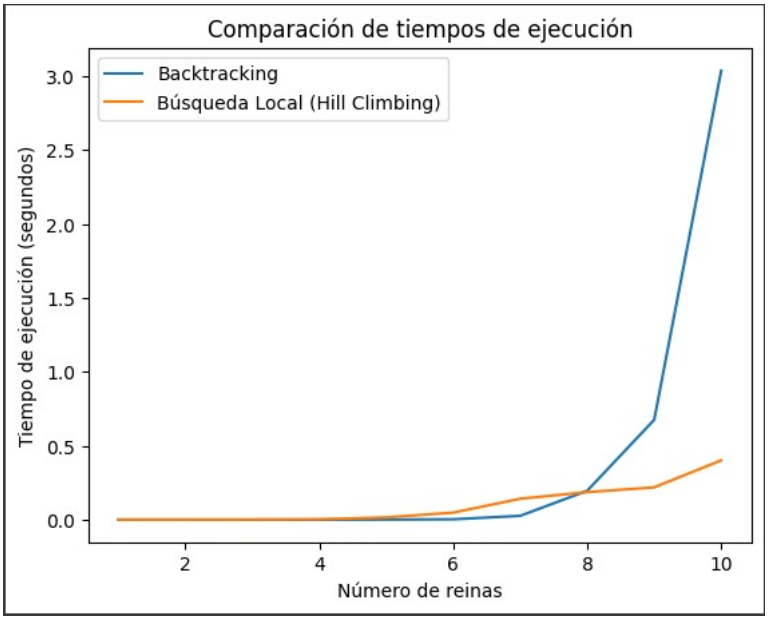

Figure 2: Grafica de resultados

Al ejecutar el codigo nos muestra esta grafica con la comparacion de los riempos de ejecucion de los posibles resultados

3 Conclusión

En este proyecto, hemos explorado a fondo la implementación y optimización del algoritmo de backtracking en Python para abordar el desafiante problema de las N reinas. A lo largo de nuestras investigaciones, no solo hemos logrado comprender en profundidad el funcionamiento fundamental del algoritmo, sino que también hemos aplicado estrategias específicas de optimización para mejorar su eficiencia.

Este trabajo tuvo enfoques que buscaron reducir el tiempo de ejecución y mejorar la escalabilidad del algoritmo, permitiendo así resolver instancias del problema de las N reinas con mayor rapidez.

Como ingenieros de sistemas, esta experiencia nos proporciona valiosas lecciones sobre la importancia de equilibrar la complejidad algorítmica con la eficiencia práctica.