TECNOLÓGICO
NACIONAL DE MÉXICO

TAREA 2

UNIDAD 2

Datos Generales:

INSTITUTO TECNOLOGICO NACIONAL DE MEXICO (CAMPUS CULIACAN)
TOPICOS DE INTELIGENCIA ARTIFICIAL
PROFESOR: ZURIEL DATHAN MORA FELIX
HORARIO:12-13

ALUMNO: BRAYANT IVAN GONZALEZ OCHOA E ISSAC PACHECO RUIZ

1. Descripción del problema

El problema de las 8 reinas consiste en colocar 8 reinas en un tablero de ajedrez de 8x8 de manera que ninguna pueda atacar a otra. Es decir, no debe haber dos reinas en la misma fila, columna o diagonal. Se trata de un problema de optimización que se puede resolver mediante varios algoritmos, incluyendo la búsqueda Tabú.

La búsqueda Tabú es una metaheurística utilizada para evitar caer en óptimos locales al restringir temporalmente ciertos movimientos considerados "prohibidos" (tabú). Se mantiene una lista Tabú que impide volver a soluciones exploradas recientemente. Esto ayuda a mejorar la eficiencia del algoritmo, evitando ciclos y explorando nuevas regiones del espacio de búsqueda.

2. Representación de P y S

Para resolver el problema con búsqueda Tabú, es importante definir claramente los elementos de nuestro espacio de búsqueda:

- Espacio de búsqueda (S): Representamos el estado como una permutación de 8 números, donde cada número indica la fila en la que se encuentra la reina en una columna dada. Esto significa que siempre habrá una reina por columna, evitando conflictos en ese sentido.
- Estado inicial (P): Se genera un estado aleatorio con 8 reinas colocadas de forma arbitraria en el tablero. Aunque no necesariamente es una solución óptima, nos permite iniciar la búsqueda desde cualquier punto del espacio.
- Función objetivo: Se mide cuántos pares de reinas se atacan entre sí. Un estado óptimo es aquel donde esta cantidad es cero.
- Movimiento: Se define como el intercambio de posiciones de dos reinas en columnas diferentes. Este cambio genera un nuevo estado que será evaluado para determinar su calidad en términos de conflictos.

3. Pseudocódigo del algoritmo

1. Generar un estado inicial aleatorio

2. Evaluar la calidad de la solución (número de conflictos entre reinas)

3. Mientras no se alcance la solución óptima y no se supere un límite de iteraciones:

   a. Generar una lista de movimientos vecinos (intercambios de posiciones de reinas)

   b. Evaluar los movimientos y seleccionar el mejor que no esté en la lista Tabú o que mejore la mejor solución encontrada

   c. Actualizar el estado y agregar el movimiento a la lista Tabú

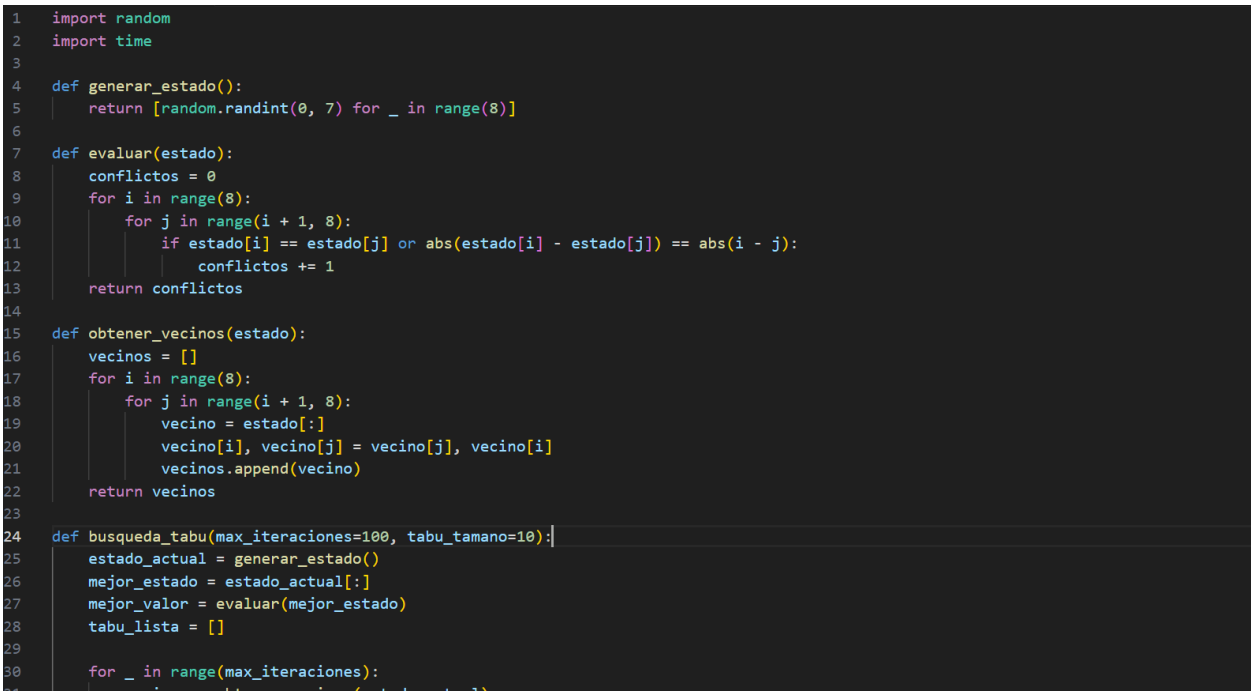
d. Si la solución es mejor que la mejor encontrada, actualizar

4. Retornar la mejor solución encontrada

4. Implementación en Python

El siguiente código en Python implementa el algoritmo de búsqueda Tabú para el problema de las 8 reinas:

En este código se muestra realizado el programa sin embargo no es el mas optimo ya que al estar corriéndolo me di cuenta que la solución mas optima que me dio fue 2 conflictos no alcanzo a llegar a la solución esperada, y la máxima fue de 7 conflictos, significa que estaba en una aparte el código atorado y no dejaba abordar correctamente cada parte del código.



```python
import random
import time

def generar_estado():
    return [random.randint(0, 7) for _ in range(8)]

def evaluar(estado):
    conflictos = 0
    for i in range(8):
        for j in range(i + 1, 8):
            if estado[i] == estado[j] or abs(estado[i] - estado[j]) == abs(i - j):
                conflictos += 1
    return conflictos

def obtener_vecinos(estado):
    vecinos = []
    for i in range(8):
        for j in range(i + 1, 8):
            vecino = estado[:]
            vecino[i], vecino[j] = vecino[j], vecino[i]
            vecinos.append(vecino)
    return vecinos

def busqueda_tabu(max_iteraciones=100, tabu_tamano=10):
    estado_actual = generar_estado()
    mejor_estado = estado_actual[:]
    mejor_valor = evaluar(mejor_estado)
    tabu_lista = []

    for _ in range(max_iteraciones):
```

```python
    for _ in range(max_iteraciones):
        vecinos = obtener_vecinos(estado_actual)
        vecinos = sorted(vecinos, key=evaluar)

        for vecino in vecinos:
            if vecino not in tabu_lista:
                estado_actual = vecino
                break

        tabu_lista.append(estado_actual)
        if len(tabu_lista) > tabu_tamano:
            tabu_lista.pop(0)

        valor_actual = evaluar(estado_actual)
        if valor_actual < mejor_valor:
            mejor_estado = estado_actual[:]
            mejor_valor = valor_actual

        if mejor_valor == 0:
            break

    return mejor_estado, mejor_valor

inicio = time.time()
solucion, conflictos = busqueda_tabu()
tiempo_total = time.time() - inicio

print("Solución encontrada:", solucion)
print("Conflictos:", conflictos)
```

```python

inicio = time.time()
solucion, conflictos = busqueda_tabu()
tiempo_total = time.time() - inicio

print("Solución encontrada:", solucion)
print("Conflictos:", conflictos)
print("Tiempo de ejecución:", tiempo_total, "segundos")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Tiempo de ejecución: 0.010869026184082031 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [6, 3, 5, 7, 1, 3, 7, 2]
Conflictos: 2
Tiempo de ejecución: 0.01811528205871582 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [2, 5, 1, 4, 7, 5, 3, 5]
Conflictos: 3
Tiempo de ejecución: 0.012291193008422852 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [1, 4, 1, 7, 0, 7, 1, 4]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Tiempo de ejecución: 0.014871597290039062 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [7, 2, 7, 7, 0, 3, 6, 4]
Conflictos: 3
Tiempo de ejecución: 0.014803886413574219 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [1, 5, 1, 1, 7, 5, 3, 1]
Conflictos: 7
Tiempo de ejecución: 0.0182340145111084 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [1, 4, 6, 0, 2, 7, 5, 3]
```

La solución mas optima es la siguiente adjunto el código con algunas pequeñas modificación para que sea mas optimo y nos de los resultados mas precisos.

8reynas.py

C: > Users > Braya > OneDrive > Desktop > 8reynas.py > ...

```python
import random
import time

def generar_estado():
    """Genera un estado inicial aleatorio para el problema de las 8 reinas."""
    return [random.randint(0, 7) for _ in range(8)]

def evaluar(estado):
    """Cuenta la cantidad de conflictos entre reinas en el tablero."""
    conflictos = 0
    for i in range(8):
        for j in range(i + 1, 8):
            if estado[i] == estado[j] or abs(estado[i] - estado[j]) == abs(i - j):
                conflictos += 1
    return conflictos

def obtener_vecinos(estado):
    """Genera vecinos modificando la posición de una única reina por columna."""
    vecinos = []
    for i in range(8):
        for nueva_fila in range(8):
            if estado[i] != nueva_fila:
                vecino = estado[:]
                vecino[i] = nueva_fila
                vecinos.append(vecino)
    return vecinos

def busqueda_tabu(max_iteraciones=500, tabu_tamano=20):
    """Implementa el algoritmo de búsqueda Tabú para resolver el problema de las 8 reinas."""
```

8reynas.py

C: > Users > Braya > OneDrive > Desktop > 8reynas.py > ...

```python
    return vecinos

def busqueda_tabu(max_iteraciones=500, tabu_tamano=20):
    """Implementa el algoritmo de búsqueda Tabú para resolver el problema de las 8 reinas."""
    estado_actual = generar_estado()
    mejor_estado = estado_actual[:]
    mejor_valor = evaluar(mejor_estado)
    tabu_lista = []

    for _ in range(max_iteraciones):
        vecinos = obtener_vecinos(estado_actual)
        vecinos = sorted(vecinos, key=evaluar)  # Ordenar por menor número de conflictos

        for vecino in vecinos:
            if vecino not in tabu_lista or evaluar(vecino) < mejor_valor:
                estado_actual = vecino
                break

        tabu_lista.append(estado_actual)
        if len(tabu_lista) > tabu_tamano:
            tabu_lista.pop(0)

        valor_actual = evaluar(estado_actual)
        if valor_actual < mejor_valor:
            mejor_estado = estado_actual[:]
            mejor_valor = valor_actual

        if mejor_valor == 0:  # Si encontramos una solución óptima, terminamos
            break
```

```python

    return mejor_estado, mejor_valor

# Ejecutar el algoritmo y medir el tiempo de ejecución
inicio = time.time()
solucion, conflictos = busqueda_tabu()
tiempo_total = time.time() - inicio

# Mostrar resultados
print("Solución encontrada:", solucion)
print("Conflictos:", conflictos)
print("Tiempo de ejecución:", tiempo_total, "segundos")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Solución encontrada: [7, 2, 7, 7, 0, 3, 6, 4]
Conflictos: 3
Tiempo de ejecución: 0.014803886413574219 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [1, 5, 1, 1, 7, 5, 3, 1]
Conflictos: 7
Tiempo de ejecución: 0.0182340145111084 segundos
PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [1, 4, 6, 0, 2, 7, 5, 3]
Conflictos: 0
Tiempo de ejecución: 0.0016579627990722656 segundos
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Braya> & C:/Users/Braya/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Braya/OneDrive/Desktop/8reynas.py
Solución encontrada: [2, 4, 1, 7, 0, 6, 3, 5]
Conflictos: 0
Tiempo de ejecución: 0.0013892650604248047 segundos
PS C:\Users\Braya>
```

5. Métricas de rendimiento

El código anterior incluye:

- Cantidad de movimientos realizados: Se puede obtener contando las iteraciones.
- Tiempo de ejecución: Se mide con time.time().
- Calidad de la solución: Se mide con la función evaluar, que cuenta conflictos.

6. Evaluación y pruebas

Para evaluar el rendimiento del algoritmo:

- Se ejecuta varias veces con estados iniciales aleatorios.
- Se registra el número de iteraciones requeridas para llegar a una solución.
- Se compara el tiempo de ejecución y la calidad de las soluciones encontradas.

Este enfoque permite analizar la eficacia del algoritmo de búsqueda Tabú para el problema de las 8 reinas.