TECNOLÓGICO
NACIONAL DE MÉXICO

TAREA N REYNAS
RECOCIDO SIMULADO

Tópicos de inteligencia artificial

Datos Generales:

INSTITUTO TECNOLOGICO NACIONAL DE MEXICO CAMPUS CULIACAN
MAESTRO: ZURIEL DATHAN MORA FELIX
HORARIO:12-13
UNIDAD 2

ALUMNOS: GONZALEZ OCHOA BRAYANT IVAN, PACHECO RUIZ ISSAC ALEJANDRO

Descripción del Problema

El problema de las 8 reinas consiste en colocar 8 reinas en un tablero de ajedrez de 8x8 de tal manera que ninguna reina amenace a otra. Esto significa que no puede haber dos reinas en la misma fila, columna o diagonal.

Representación de P y S

- **P (Problema)**: El problema se representa como un tablero de 8x8, donde cada celda puede estar vacía o contener una reina.
- **S (Solución)**: Una solución es una configuración del tablero donde se colocan 8 reinas sin que se amenacen entre sí.

Propuesta de Algoritmo en Pseudocódigo

1. Inicializar una solución inicial S (puede ser aleatoria o una configuración específica).
2. Inicializar la lista Tabú como vacía.
3. Definir el tamaño máximo de la lista Tabú (TamañoTabú).
4. Definir el número máximo de iteraciones (MaxIteraciones).
5. Para cada iteración hasta MaxIteraciones:
   a. Generar todos los vecinos de S (movimientos posibles de las reinas).
   b. Evaluar cada vecino y seleccionar el mejor que no esté en la lista Tabú.
   c. Si el mejor vecino es mejor que S, actualizar S.
   d. Agregar el movimiento a la lista Tabú.
   e. Si la lista Tabú excede TamañoTabú, eliminar el elemento más antiguo.
6. Devolver la solución S.

Código en Python

```python
import random
import time

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

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

def generar_vecinos(tablero):
    vecinos = []
    for i in range(8):
        for j in range(8):
            if tablero[i] != j:
                vecino = list(tablero)
                vecino[i] = j
                vecinos.append(vecino)
    return vecinos

def busqueda_tabu(max_iteraciones, tamano_tabu):
    tablero = inicializar_tablero()
    mejor_tablero = list(tablero)
    lista_tabu = []
```

```
    movimientos = 0

    inicio = time.time()

    for iteracion in range(max_iteraciones):
        vecinos = generar_vecinos(tablero)
        mejor_vecino = None
        mejor_conflictos = float('inf')

        for vecino in vecinos:
            if vecino not in lista_tabu:
                conflictos = calcular_conflictos(vecino)
                if conflictos < mejor_conflictos:
                    mejor_conflictos = conflictos
                    mejor_vecino = vecino

        if mejor_vecino is None:
            break

        tablero = mejor_vecino
        movimientos += 1

        if calcular_conflictos(tablero) < calcular_conflictos(mejor_tablero):
            mejor_tablero = list(tablero)

        lista_tabu.append(list(tablero))
```

```
        lista_tabu.append(list(tablero))
        if len(lista_tabu) > tamano_tabu:
            lista_tabu.pop(0)

    fin = time.time()
    tiempo_ejecucion = fin - inicio

    return mejor_tablero, movimientos, tiempo_ejecucion

# Parámetros
max_iteraciones = 1000
tamano_tabu = 10

# Ejecución
mejor_tablero, movimientos, tiempo_ejecucion = busqueda_tabu(max_iteraciones, tamano_tabu)

# Resultados
print("Mejor tablero encontrado:", mejor_tablero)
print("Número de movimientos:", movimientos)
print("Tiempo de ejecución:", tiempo_ejecucion, "segundos")
```