Universidad Nacional Autónoma de México

Facultad de Ciencias

Complejidad Computacional

Práctica 3
Metaheurísticas

Lucy Gasca Soto
Brenda Margarita Becerra Ruiz

Autor
Delgado Díaz Hermes Alberto
319258613

30 de mayo de 2025

Índice

1. Problema

El problema de las `N-Reinas` consiste en colocar n reinas en un tablero de ajedrez de $n \times n$ de forma que ninguna se ataque(colisione) entre si.
En la implementación, se utiliza una representación de permutación donde:

- Cada posición de la lista representa una columna del tablero.
- El valor en cada posición representa la fila donde se coloca la reina.

Se implementaron dos metaheurísticas para resolver el problema:

1. ***Algoritmos Genéticos.***

   Es un algoritmo inspirado en la evolución natural. Utilizan operadores genéticos(selección, cruce y mutación) para evolucionar una población de soluciones candidatas hacia mejores soluciones.

2. ***Búsqueda Tabú.***

   La búsqueda tabú es una metaheurística que utiliza estructuras de memoria para guiar la búsqueda local y evitar ciclos. Mantiene una lista de movimientos prohibidos (tabú) para diversificar la exploración.

2. Algoritmo Genético

Función de evaluación

El algoritmo utiliza una función llamada `fitness` que cuenta cuantas colisiones hay en la posible solución.

```
def fitness (tablero):
```

Dos reinas están en la misma diagonal si la diferencia absoluta de sus filas es igual a la diferencia absoluta de sus columnas. Una solución óptima tiene evaluación = 0 (sin colisiones).

2.1. Generación de población inicial

```
def generar_población (n,tam, base = None):
```

- Crea una población de *tam* individuos.
- Cada individuo es una permutación aleatoria de números 0 a $n - 1$.
- Si se proporciona una *base*, se genera diversidas a partir de ella.

2.2. Selección por Ruleta

```
def seleccion_ruleta (poblacion):
```

- Selecciona padres con probabilidad proporcional a su aptitud.
- Utiliza fitness inverso: $1/(1 + fitness(indiduo))$
- Individuos con menos colisiones tienen mayor probabilidad de ser seleccionados.

2.3. Cruce

```
def cruce (ind1, ind2):
```

- Implementa cruce de orden(*Order Crossover*).
- Mantiene parte del primer padre y completa con elementos del segundo.
- Preserva la propiedad de permutación(no hay reinas duplicadas).

2.4. Mutación

```
def mutacion (ind, tasa=0.1):
```

- Realiza mutación por intercambio con probabilidad del 10 %.
- Intercambia aleatoriamente dos posiciones del individuo.
- Mantiene la validez de la permutación.

2.5. Algoritmo Principal

```
def algoritmo_genetico (n=8, generaciones = 200, tam_poblacion = 50, inicial = None):
```

Parámetros:

- n: Tamaño del tablero (8 por defecto).
- generaciones: Número máximo de iteraciones(200).
- tam_poblacion: Tamaño de la población(50).
- inicial: Configuración inicial opcional.

Proceso:

1. Generar población inicial.
2. Para cada generación:
   - Seleccionar padres mediante ruleta.
   - Aplicar cruce para generar descendientes.
   - Aplicar mutación.
   - Reemplazar población completa.
   - Evaluar y mantener el mejor individuo.
3. Terminar si se encuentra una solución óptima o se agotan las generaciones.

3. Búsqueda Tabú

3.1. Función de evaluación.

El algoritmo utiliza una función llamada `evaluar` que cuenta cuantas colisiones hay en la posible solución. En este caso, se utiliza la función `fitness` ya que la función actúa de la misma manera.

3.2. Generación de Vecindario

```
def generar_vecindario (sol):
```

- Genera todos los vecinos posibles mediante intercambios de pares.
- Para un tablero de tamaño n, genera $n(n-1)/2$ vecinos.
- Cada vecino se obtiene intercambiando dos reinas de posición.

3.3. Lista Tabú

- Almacena los movimientos (intercambios) realizados recientemente.
- Tamaño controlado por tabu_tenure (5 por defecto).
- Funciona como memoria a corto plazo para evitar ciclos.

Criterio de Aspiración
El algoritmo permite movimientos tabú si mejoran la mejor solución conocida:

```
if movimiento not in lista_tabu or evaluar(vecino) <evaluar(mejor_sol):
```

3.4. Algoritmo Principal

```
def busqueda_tabu (n=8, max_iter = 100, tabu_tenure = 5, inicial = None):
```

Parámetros:

- n: Tamaño del tablero (8 por defecto).
- max_iter: Número máximo de iteraciones(100).
- tabu_tenure Tamaño de la lista tabú(5).
- inicial: Configuración inicial opcional.

Proceso:

1. Inicializar con solución aleatoria o proporcionada.
2. Para cada iteración:
    - Generar vecindario completo.
    - Ordenar vecinos por calidad(fitness).
    - Seleccionar el mejor vecino no tabú.
    - Actualizar lista tabú.
    - Actualizar mejor solución global.
3. Terminar si se encuentra solución óptima o se agotan las iteraciones.

4. Pruebas realizadas

En la práctica se realizaron tres pruebas con distintos valores de n.

4.1. N = 8

Tablero aleatorio

Se genera un tablero aleatorio de 8×8 para encontrarle una solución.

```
=== Prueba con n = 8 ===

--- Tablero Inicial Aleatorio ---
Configuración: [0, 4, 1, 2, 6, 5, 3, 7]
Q . . . . . . .
. . Q . . . . .
. . . Q . . . .
. . . . . . Q .
. Q . . . . . .
. . . . . Q . .
. . . . Q . . .
. . . . . . . Q
```

Algoritmo genético

```
--- Algoritmo Genético ---
Solución: [5, 0, 4, 1, 7, 2, 6, 3]
Colisiones: 0
Iteración encontrada: 20
Iteración máxima permitida: 200
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . . . . . . Q
. . Q . . . . .
Q . . . . . . .
. . . . . Q . .
. . . . Q . . .
```

Búsqueda Tabú

```
--- Búsqueda Tabú ---
Solución: [4, 6, 0, 2, 7, 5, 3, 1]
Colisiones: 0
Iteración encontrada: 4
Iteración máxima permitida: 100
. . Q . . . . .
. . . . . . . Q
. . . Q . . . .
. . . . . Q . .
Q . . . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . Q . . .
```

4.2. N = 10

Tablero aleatorio

Se genera un tablero aleatorio de 10×10 para encontrarle una solución.

```
=== Prueba con n = 10 ===

--- Tablero Inicial Aleatorio ---
Configuración: [2, 9, 4, 0, 5, 8, 7, 1, 6, 3]
. . . Q . . . . . .
. . . . . . . Q . .
Q . . . . . . . . .
. . . . . . . . . Q
. . Q . . . . . . .
. . . . Q . . . . .
. . . . . . . . Q .
. . . . . . Q . . .
. . . . . Q . . . .
. Q . . . . . . . .
```

Algoritmo genético

```
--- Algoritmo Genético ---
Solución: [7, 1, 4, 0, 8, 3, 9, 6, 2, 5]
Colisiones: 0
Iteración encontrada: 11
Iteración máxima permitida: 200
. . . Q . . . . . .
. Q . . . . . . . .
. . . . . . . . Q .
. . . . . Q . . . .
. . Q . . . . . . .
. . . . . . . . . Q
. . . . . . . Q . .
Q . . . . . . . . .
. . . . Q . . . . .
. . . . . . Q . . .
```

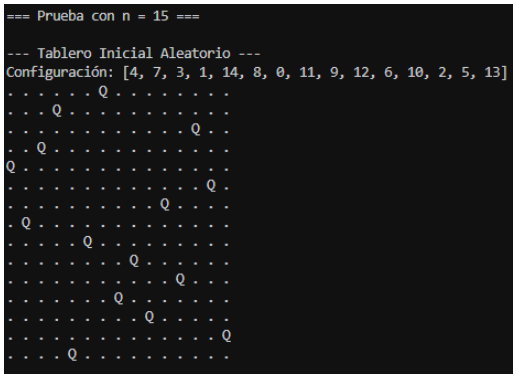
Búsqueda Tabú

```
--- Búsqueda Tabú ---
Solución: [8, 4, 7, 0, 2, 5, 1, 6, 9, 3]
Colisiones: 0
Iteración encontrada: 8
Iteración máxima permitida: 100
. . . Q . . . . . .
. . . . . . Q . . .
. . . . Q . . . . .
. . . . . . . . . Q
. Q . . . . . . . .
. . . . . Q . . . .
. . . . . . . Q . .
. . Q . . . . . . .
Q . . . . . . . . .
. . . . . . . . Q .
```

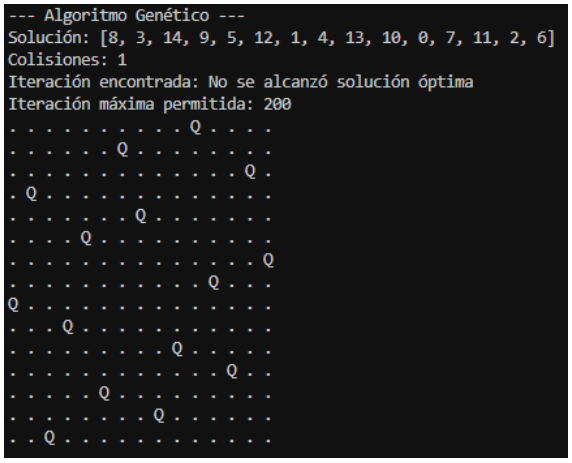
4.3. N = 15

Tablero aleatorio

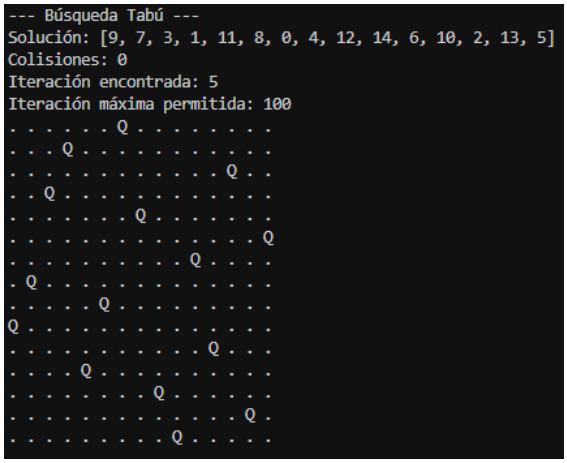
Se genera un tablero aleatorio de 15×15 para encontrarle una solución.



Algoritmo genético



Búsqueda Tabú



5. Código

Se realizo la práctica en `Python`
Ejecución:

```
?- Python NReinasProblem.py
```

El código incluye tres ejecuciones con el tablero tamaño
$n = [8, 10, 15]$