



**UNIVERSIDADE ESTADUAL DA PARAÍBA**  
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

DJHONATAH WESLEY C ALVES

**Relatório**  
**Problema das N-Rainhas com bloqueio**

CAMPINA GRANDE

2025

## ESTRUTURA DE DADOS E ABORDAGEM

Através de pesquisas, foi demonstrado que a solução mais eficiente em relação a eficiência traduzida em otimização, rapidez e robustez foi o Algoritmo Genético, chegamos à seguinte conclusão mediante a pesquisas que utilizavam de outros algoritmos como o algoritmo A\* (A-Estrela), onde foi observado uma dificuldade maior em resolver o problema em questão, sabendo que ele funciona tentando expandir todos os caminhos possíveis, linha por linha, posicionando uma rainha por vez, ele também gasta muito tempo tentando caminhos mortos. O objetivo principal da atividade é posicionar todas as rainhas sem que elas se ataquem, e sem ocupar casas bloqueadas, de forma que para encontrar uma configuração válida de  $n$  rainhas em um tabuleiro  $n \times n$ . Utilizando do Algoritmo Genético foi aplicado, para diversos tamanhos de  $n$ : **1, 2, 5, 8, 9, 12, 16 e 20**. Os testes foram executados em um tabuleiro gerado entre **7% e 13%** de casas bloqueadas, evitando que houvesse uma linha inteira bloqueada. Através de pesquisas também foi incorporado mecanismos de elitismo e mini reparos para otimizar ainda mais a performance e reduzir os conflitos.

### Descrição da Política de Expansão

Como o Algoritmo Genético **não funciona como uma busca em árvore**, ele não usa heurística explícita, mas sim utiliza de operadores evolutivos:

1. **Seleção:** por torneio ( $k = 3$ ), onde os melhores indivíduos são escolhidos para que a partir deles haja indivíduos cada vez melhores.

```
1 def tournament_selection(pop, fit_vals, k=3):
2     competidores = random.sample(range(len(pop)), k)
3     vencedor = max(competidores, key=lambda i: fit_vals[i])
4     return pop[vencedor]
5
```

2. **Crossover:** Ele **combina partes de dois indivíduos (pais)** para gerar novos indivíduos (filhos), com a ideia de **recombinar boas características** dos pais.

```
1 def crossover(pai1, pai2, pc):
2     n = len(pai1)
3     if random.random() < pc:
4         ponto = random.randint(1, n-1)
5         return pai1[:ponto] + pai2[ponto:], pai2[:ponto] + pai1[ponto:]
6     return pai1[:], pai2[:]
```

3. **Mutação:** com probabilidade 0.1 por gene, sorteando uma nova coluna válida (não bloqueada).

```
1 def mutacao(individuo, bloqueios, pm):
2     n = len(individuo)
3     for i in range(n):
4         if random.random() < pm:
5             individuo[i] = random.choice([c for c in range(n) if (i, c) not in bloqueios])
6     return individuo
```

4. **MiniRepair:** É uma técnica usada em algoritmos genéticos para corrigir parcialmente indivíduos da população que possuem muitos conflitos (ou seja, soluções ruins)
5. **Elitismo:** É uma estratégia nos algoritmos genéticos onde os melhores indivíduos da geração atual são automaticamente copiados para a próxima geração sem alterações.

```
1 pop_sorted = [ind for _, ind in sorted(zip(fit_vals, pop), key=Lambda x: x[0], reverse=True)]
2 nova_pop = pop_sorted[:elitismo_k] # preserva elites
```

6. **Cr terios de parada:** Caso haja a solu  o encontrada ou se atingir o n vel m ximo de gera  es = 2000.

## Ambiente de Execu  o

Os testes do algoritmo gen tico foram realizados em um computador com as seguintes configura  es:

- **Processador (CPU):** AMD Ryzen 5 5600G @ 3.90GHz (6 n cleos, 12 threads)
- **Mem ria RAM:** 16 GB DDR4
- **Sistema Operacional:** Windows 11 64 bits
- **Vers o do Python:** 3.10.11
- **Editor/IDE utilizado:** VS Code
- **Seed aleat ria usada:** 42 (para garantir reprodutibilidade dos testes)

## Resultados e Discuss o

N	SOLU��O ENCONTRADA	GERA��ES USADAS	TEMPO (MS)
1	SIM	1	0.17
2	N�O	2000	5469.32
5	SIM	1	1.01
8	SIM	2	31.67
9	SIM	2	42.32
10	SIM	2	55.05
12	SIM	3	174.44
16	SIM	57	10533.97
20	SIM	175	61804.49

Foi observado que mesmo com 2000 gera  es para o tamanho  $n=2$  n o foi poss vel encontrar solu  o, o que   consistente com a teoria: n o h  solu  o

para 2 rainhas que não se ataquem em um tabuleiro  $2 \times 2$ . O algoritmo genético se mostrou robusto e completo, especialmente com o uso de elitismo e minireparo, permitindo encontrar soluções rapidamente para tamanhos de tabuleiro grandes  $n=20$ , por exemplo. A abordagem evolutiva permite explorar uma ampla diversidade de soluções mesmo em cenários com restrições significativas. Os testes demonstraram desempenho consistente e boa escalabilidade.