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 x 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.

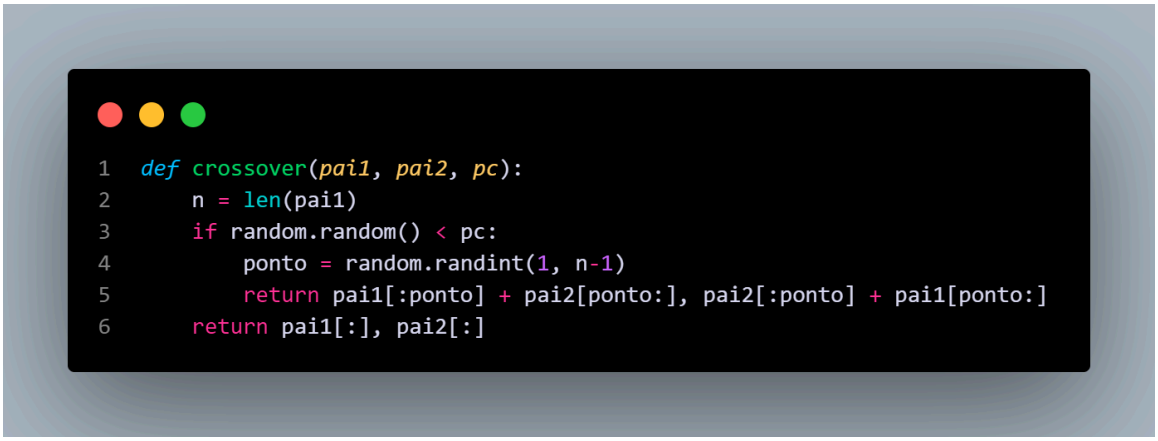
Descrição da Política de Expansão

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 individuos são escolhidos para que a partir deles haja indivíduos cada vez melhores.

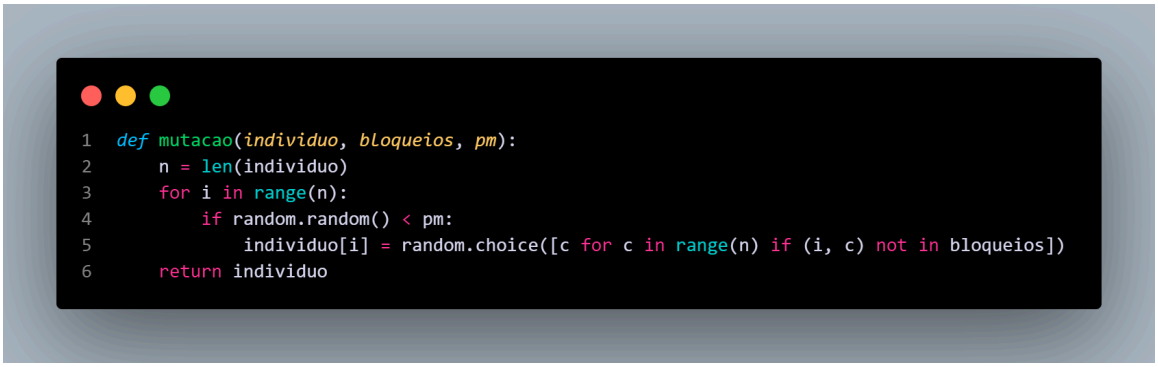
```python
def tournament_selection(pop, fit_vals, k=3):
    competidores = random.sample(range(len(pop)), k)
    vencedor = max(competidores, key=lambda i: fit_vals[i])
    return pop[vencedor]

```

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



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

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



```python
def mutacao(individuo, bloqueios, pm):
    n = len(individuo)
    for i in range(n):
        if random.random() < pm:
            individuo[i] = random.choice([c for c in range(n) if (i, c) not in bloqueios])
    return individuo
```

4. **Critérios de parada**: Caso haja a solução encontrada ou se atingir o nível máximo de gerações = 2000.

Ambiente de Execução

Os testes do algoritmo genético foram realizados em um computador com as seguintes configurações:

- **Processador (CPU):** AMD Ryzen 5 5600G @ 3.90GHz (6 núcleos, 12 threads)
- **Memória RAM:** 16 GB DDR4
- **Sistema Operacional:** Windows 11 64 bits
- **Versão do Python:** 3.10.11
- **Editor/IDE utilizado:** VS Code
- **Seed aleatória usada:** 42 (para garantir reprodutibilidade dos testes)

Resultados e Discussão

N	SOLUÇÃO ENCONTRADA	GERAÇÕES USADAS	TEMPO (MS)
1	SIM	1	0.17
2	NÃO	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 gerações para o tamanho n=2 não foi possível encontrar solução, o que é consistente com a teoria: não há solução para 2 rainhas que não se ataquem em um tabuleiro 2x2. 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.