Análise Comparativa de Algoritmos de Busca Heurística para o Problema das N-Rainhas

Aluna: Fernanda Lima de Souza

Resumo

Este trabalho investiga a eficácia de três algoritmos de busca heurística: Hill-Climbing, Simulated Annealing e Algoritmo Genético na resolução do problema das N-Rainhas para tabuleiros de dimensões N = {32, 64, 128}. Cada algoritmo foi implementado, testado e avaliado com base na qualidade da solução, custo computacional e confiabilidade. Os resultados experimentais demonstraram uma superioridade para com Simulated Annealing, que mostrou o melhor equilíbrio entre tempo de execução e taxa de sucesso, encontrando soluções ótimas de forma consistente. O Hill-Climbing, embora rápido, mostrou-se ineficaz devido à sua vulnerabilidade a mínimos locais, enquanto o Algoritmo Genético, apesar de sua robustez teórica, revelou-se computacionalmente caro e sensível à parametrização para este problema específico.

***Palavras-chave**: N-Rainhas, Hill-Climbing, Simulated Annealing, Algoritmo Genético, Busca Heurística, Otimização.*

---

1. Introdução

O problema abordado neste trabalho é o das **N-Rainhas,** um problema combinatório clássico baseado em jogo de xadrez e tal problema consiste em posicionar **N-Rainhas** em um t**abuleiro N × N** de tal maneira que nenhuma delas se ataquem, garantindo que:

- Não pode haver mais de uma rainha na **mesma linha**.
- Nem na **mesma coluna**.
- E também não podem estar na **mesma diagonal**.

Este problema é muito utilizado como *bechmark* em Inteligência Artificial, pois é um desafio clássico de busca em espaços de solução complexos. À medida que o número de N aumenta, o número de possíveis configurações cresce **exponencialmente**, tornando mais difícil encontrar uma solução ideal. Além disso, o problema das N-Rainhas testa a capacidade de algoritmos em:

- **Evitar mínimos locais**, ou seja, soluções que parecem boas, mas não são ideais;
- **Explorar eficientemente o espaço de busca**, que é o conjunto de todas as possíveis configurações de rainhas no tabuleiro;
- **Encontrar soluções ótimas ou próximas disso** com rapidez, mesmo em situações onde há muitas possibilidades erradas no caminho.

Este trabalho tem como objetivo a implementação, execução de testes e comparação de resultados dos algoritmos **Hill-Climbing**, **Simulated Annealing** e **Algoritmo Genético** a

modo de resolver o problema proposto das N-Rainhas, além de analisar o desempenho de cada abordagem na resolução do problema.

2. Modelagem e Metodologia

Nesta seção, é descrito como o problema das N-Rainhas foi modelado e como cada algoritmo foi implementado, incluindo os parâmetros usados e suas justificativas.

Definições Gerais - Comuns a todos os algoritmos

Para garantir eficiência, uma modelagem otimizada foi adotada:

- O tabuleiro é representado por uma **lista de inteiros**, onde o índice representa a **coluna** e o valor armazenado representa a **linha** da rainha naquela coluna. Por exemplo, o vetor [0, 2, 4, 1] representa um tabuleiro 4×4 com rainhas nas posições (0,0), (1,2), (2,4) e (3,1).

- A **função de custo** utilizada calcula o **número de pares de rainhas que se atacam**. Quanto menor o valor do custo, melhor a solução.

- A **função de fitness**, usada no Algoritmo Genético, foi definida como 1 / (1 + custo), de forma que soluções com custo 0 (ótimas) tenham fitness igual a 1.

a. Hill Climbing

- **Definição de Vizinhança:** Um vizinho é gerado **movendo uma única rainha para outra linha da mesma coluna**, mantendo todas as outras na mesma posição. Isso garante que a busca explore o espaço local ao redor do estado atual.

- **Critério de escolha:** Em cada iteração, o algoritmo escolhe o **vizinho com menor custo** (caso exista). Se nenhum vizinho for melhor que o estado atual, considera-se que o algoritmo chegou a um **mínimo local**.

- **Parâmetros utilizados:**
    - max_interacoes = 1000: evita loops infinitos em platôs.
    - Inicialização aleatória do tabuleiro.

b. Simulated Annealing

- **Definição de Vizinhança:** O vizinho é gerado **aleatoriamente**, selecionando uma coluna ao acaso e movendo sua rainha para uma linha diferente. Isso permite maior diversidade nos estados explorados.

- **Critério de aceitação:** Se o novo estado tem **custo menor**, ele é aceito. Caso contrário, pode ser aceito com uma **probabilidade que depende da diferença de custo e da temperatura atual** (math.exp(-delta_custo / temperatura)), permitindo a fuga de mínimos locais.

- **Parâmetros utilizados:**
    - Temperatura inicial: 1000
    - Temperatura final: 0.0001
    - Taxa de resfriamento: 0.99
    - Iterações por temperatura: 100
    - Esses valores foram escolhidos para permitir uma boa exploração inicial e resfriamento gradual, balanceando exploração e convergência.

c. Algoritmo Genético

- **Representação:** Cada indivíduo é um tabuleiro representado como uma lista (cromossomo). A população é composta por um conjunto de 300 indivíduos.

- **Operadores Genéticos:**
    - **Seleção:** foi utilizado **torneio entre três indivíduos**, onde o indivíduo de maior fitness é escolhido como pai.
    - **Crossover:** cruzamento de um ponto, onde uma parte do pai 1 é combinada com o restante do pai 2.
    - **Mutação:** altera aleatoriamente a posição de uma rainha (uma linha aleatória em uma coluna aleatória).

- **Parâmetros utilizados:**
    - Tamanho da população: 300
    - Número de gerações: 2500
    - Taxa de crossover: 0.8
    - Taxa de mutação: 0.2

Estes valores foram definidos empiricamente, visando garantir diversidade e permitir convergência dentro de um número razoável de gerações.

3. Resultados Obtidos

Os experimentos foram realizados com valores de N = 32, 64 e 128. Para cada algoritmo foram executados 5 testes pelo valor de N. Os testes ocorreram em ambiente Linux com um processador AMD Ryzen 7 5800 h, 8GB de RAM, GPU GeForce GTX 1650, utilizando a linguagem Python 3.10.12. A seguir são apresentadas as tabelas com as médias dos resultados.

Tabela de Resultados Finais: **Hill-Climbing**

N (Tamanho do Tabuleiro)	Custo Médio Final	Tempo Médio (s)	Confiabilidade (Sucessos)
32	3.20	0.69 segundos	0/5 (0%)
64	3.20	23.16 segundos	0/5 (0%)
128	5.40	1060.90 segundos (~17.7 minutos)	0/5 (0%)

Tabela de Resultados Finais: **Simulated Annealing**

N (Tamanho do Tabuleiro)	Custo Médio Final	Tempo Médio (s)	Confiabilidade (Sucessos)
32	0.00	4.90 segundos	5/5 (100%)
64	0.00	25.53 segundos	5/5 (100%)
128	0.60	116. 54 segundos (~1.9 minutos)	2/5 (40%)

Tabela de Resultados Finais: **Algoritmo Genético**

N (Tamanho do Tabuleiro)	Custo Médio Final	Tempo Médio (s)	Confiabilidade (Sucessos)
32	0.60	28.93 segundos	2/5 (40%)
64	1.40	156.30 (~2.6 min)	0/5 (0%)
128	2.60	7080.35 (~118 min)	0/5 (0%)

Uma análise inicial dos dados consolidados aponta para uma superioridade marcante do Simulated Annealing em termos de confiabilidade e um trade-off complexo entre os três métodos no que tange ao custo computacional.

4. Discussão sobre o Comportamento dos Métodos

A análise comparativa revela as características distintas de cada algoritmo frente ao desafio de escalabilidade e complexidade do problema.

O Hill-Climbing, se mostrou limitado e com desempenho razoável para tabuleiros pequenos, e menos eficiente ao escalar para valores maiores de N. Seu comportamento guloso o impede de escapar de mínimos locais, o que o torna ineficaz na maioria dos testes realizados. Mesmo com inicializações aleatórias e 1000 iterações por tentativa, não foi capaz de encontrar nenhuma solução ótima, independentemente do tamanho do tabuleiro.

Já o **Simulated Annealing** provou ser o algoritmo mais equilibrado. Sua capacidade de aceitar soluções piores no início, controlada pela temperatura, foi determinante para escapar de armadilhas locais e explorar melhor o espaço de busca. Obteve 100% de confiabilidade para N=32 e N=64, e mesmo para N=128, onde o espaço de busca se torna massivo, ainda conseguiu encontrar soluções ótimas em 40% das execuções. Além disso, apresentou tempos significativamente menores do que o Algoritmo Genético.

E por fim, **o Algoritmo Genético**, embora poderoso, teve dificuldades em convergir para soluções ótimas, especialmente para N maiores. Mesmo com o aumento dos parâmetros e uma população relativamente grande (300 indivíduos), número elevado de gerações (2500), e taxa de mutação 0.2 sua taxa de sucesso foi baixa. Para N=128, o tempo médio de quase 2 horas o torna o método menos prático. A performance inferior sugere que, para este problema, a estratégia de busca do AG, baseada em recombinação (crossover), pode ser menos eficiente que a busca local perturbativa do Simulated Annealing. O crossover de um ponto, em particular, pode ser disruptivo, quebrando boas soluções parciais e exigindo mais tempo para a evolução se recuperar.

Os dados expõem um claro *trade-off*. O Hill-Climbing oferece velocidade em troca de qualidade. O Algoritmo Genético oferece robustez teórica, mas a um custo computacional elevado e alta sensibilidade aos parâmetros e operadores. O **Simulated Annealing emergiu como a solução de melhor custo-benefício**, combinando alta confiabilidade com um tempo de execução razoável e escalável.

5. Conclusão

Este estudo realizou a implementação e comparação de três abordagens heurísticas para o problema das N-Rainhas. A análise experimental demonstrou que a escolha do algoritmo impacta a capacidade de encontrar soluções ótimas de maneira eficiente.

O **Simulated Annealing destacou-se como o método mais eficaz**, tendo equilíbrio entre tempo de execução e confiabilidade. Sua capacidade de escapar de mínimos locais, conjuntamente a uma complexidade computacional gerenciável, o fez superior nos cenários testados. O Algoritmo Genético, embora tenha seu potencial, exigiu parametrização cuidadosa e ainda assim foi superado pelo SA em questão de custo-benefício. O Hill-Climbing serviu como uma importante base de comparação, mostrando as limitações de uma abordagem totalmente gulosa.

O estudo reforça que não há um "melhor algoritmo" universal, porém uma escolha mais adequada às características do problema. O problema das N-Rainhas continua, assim, sendo uma excelente plataforma para o estudo e aprofundamento de técnicas de otimização em Inteligência Artificial.