CENTRO DE TECNOLOGIA – CTC

DEPARTAMENTO DE INFORMÁTICA – DIN

BACHARELADO EM INFORMÁTICA

JOHN WILLIAM VICENTE – RA: 118237

NAYANE BATISTA COSTA – RA: 117189

PROJETO PRÁTICO EM INTELIGÊNCIA ARTIFICIAL: PROBLEMA DAS N-RAINHAS E COMO RESOLVÊ-LO ATRAVÉS DA IMPLEMENTAÇÃO DE ALGORITMOS

MARINGÁ,

2024

RESUMO

Este artigo aborda a aplicação prática de técnicas de Inteligência Artificial na resolução do clássico problema das N-Rainhas, que consiste em posicionar N rainhas em um tabuleiro de xadrez NxN de forma que nenhuma delas esteja em uma diagonal ou linha da mesma coluna, ou seja, sem que elas se ataquem mutuamente.

O projeto implementou quatro técnicas que envolviam algoritmos específicos conhecidos na literatura para resolver o problema com a linguagem de programação Java. A pesquisa demonstrou como podemos adaptar determinados algoritmos para lidarmos com o problema das N-Rainhas, oferecendo diferentes tipos de soluções. Ao longo do trabalho, são discutidos os desafios específicos associados ao problema, as escolhas de implementação e as métricas de avaliação de desempenho dos algoritmos utilizados.

1. INTRODUÇÃO

O presente trabalho foi elaborado a partir do cenário do problema das N-Rainhas, onde a partir deste, a equipe objetivou a elaboração dos algoritmos de Backtracking, Minimum Conflicts, aplicação da técnica de Propagation e a construção de um algoritmo genético para o encontro da solução do referido problema, além da compreensão teórica do N-Rainhas.

O escopo deste trabalho abrange a apresentação do problema e análise detalhada dos algoritmos desenvolvidos, através dos testes e validações quanto a eficiência das abordagens propostas. É importante ressaltar que este estudo não se limitou a implementação de algoritmos, mas também incorporou uma base teórica sólida adquirida por revisão bibliográfica.

A pesquisa é estruturada em seções com detalhamentos acerca da fundamentação proposta pela dupla durante a execução do projeto, aplicação de experimentos práticos, análise dos resultados obtidos e, por fim, as conclusões, além de exercitar conceitos práticos da implementação de problemas clássicos, juntamente com a carga de teoria trazida por trabalhos em Inteligência Artificial e os algoritmos que integram a área estudada.

2. FUNDAMENTAÇÃO E METODOLOGIA

Conforme os apontamentos feitos na introdução deste relatório, o trabalho prático se subdividiu na implementação de quatro algoritmos no intuito de compararmos a eficiência dos mesmos na resolução de um mesmo problema, no caso do presente documento, o problema das N-Rainhas.

O problema de N-Rainhas é um desafio clássico que consiste em posicionar uma quantidade N de rainhas em um tabuleiro de xadrez NxN, sendo uma generalização do problema das 8 Rainhas – dispor 8 rainhas em um tabuleiro 8x8 – sem que elas se ataquem reciprocamente, seguindo as regras do jogo de xadrez. O problema das N-Rainhas possui solução para todo $N \geq 4$.

Os algoritmos mais comuns utilizados para abordar o problema das N-Rainhas incluem o algoritmo de *Backtracking*, que explora todas as possibilidades de posicionamento das rainhas, o algoritmo genético, que utiliza conceitos trazidos das Ciências Biológicas e evolução das espécies, e algoritmos com heurísticas, que aplicam estratégias inteligentes para redução da complexidade computacional.

Quanto às aplicações práticas do problema das N-Rainhas, podemos correlacionar aos problemas de alocação de tarefas, planejamento de uso de recursos e posicionamento logístico.

2.1. Algoritmo *Backtracking* (Retrocesso)

Também conhecido como uma forma refinada do algoritmo de Força Bruta, Busca Completa ou Enumeração Exaustiva, o *Backtracking* consiste em métodos para vasculhar o espaço de busca por completo ou em partes buscando a solução desejada. Um exemplo de aplicação do *Backtracking* é o problema clássico do Caixeiro Viajante.

No caso do problema das N-Rainhas, há a imposição de restrições e condições no *Backtracking*, tratando-se assim de uma classe de problemas que é conhecida como *Constraint Satisfaction Problem* (CSP), traduzida para Problemas de Satisfação de Restrições (PSR).

O *Constraint Satisfaction Problems* é consistido de um conjunto de variáveis que podem assumir valores dentro de um dado domínio e um conjunto de restrições que especificam propriedades da solução. É essencial no *Backtracking* que se defina um espaço de

solução para o problema que inclua a solução ótima e que possa ser pesquisada de forma organizada.

Sem a programação das *constraints*, os *backtrackings* sempre tenderão a realizar buscas exaustivas, com grandes chances de ocorrência de explosões combinatórias, além da necessidade de grandes quantidades de memórias disponíveis para eles. Um ponto positivo dessa abordagem, é a simplicidade de implementação onde outras formas de resolução seriam mais complexas.

2.2. Algoritmo *Minimum Conflicts*

Esse algoritmo é outro exemplo muito usado para a solução do N-Rainhas. Trata-se de um algoritmo de busca para resolução de problemas de satisfação de restrições (CSP iterativo).

O *Minimum Conflicts* (*min-conflicts*) funciona a partir de uma atribuição inicial de valores às variáveis do problema. A cada iteração, o algoritmo seleciona aleatoriamente uma variável com conflitos e atribui a ela o valor que minimiza o número desses conflitos. Se houver mais de um valor com o mesmo número mínimo de conflitos, um deles é escolhido aleatoriamente.

O *min-conflicts* pode ser visto como uma heurística de reparo que escolhe o estado com o menor número de conflitos. Uma boa atribuição inicial pode ser crucial para se aproximar rapidamente de uma solução.

2.3. Uso da técnica de *Propagation*

O termo "*Propagation*" é genérico dentro da I.A e se refere ao processo de propagação da informação como uma consequência da verificação de arco-consistência de um problema de busca. Quando um valor é eliminado, outros podem se tornar inconsistentes e terem que ser eliminados também, assim como uma onda que se propaga e deixa as escolhas cada vez mais restritas.

Através do processo de propagação é possível simplificar e otimizar o funcionamento dos algoritmos empregados em determinado problema. No problema do N-Rainhas, o emprego dessa técnica possibilitou com que os algoritmos de *Backtracking* e *min-conflicts* encontrassem informações de forma mais rápida, evitando ter que realizar retrabalhos para calcular o número de conflitos em determinado instante da execução do programa.

2.4. Algoritmo Genético

Os algoritmos genéticos buscam resolver problemas computacionais de otimização simulando o comportamento da seleção natural de candidatas à soluções, encontrado na natureza e descrito primeiramente por Charles Darwin.

Nessa teoria, dentro de uma população de indivíduos há uma disputa de recursos e uma necessidade de sobrevivência em um determinado ambiente, onde existe uma tendência para que indivíduos mais aptos consigam reproduzir-se deixando suas características genéticas para as gerações futuras. Com o tempo, as gerações de indivíduos começam a se especializar e se tornarem mais aptas em seu ambiente, seja através do recebimento do material genético das gerações pais, quanto por mutações geradas de forma natural.

Essa solução pode ser aplicada ao N-Rainhas consistindo uma forma interessante de busca local, a fim de encontrar uma solução que atenda todas as restrições do problema.

3. EXPERIMENTOS

Para este projeto, foram realizadas quatro diferentes implementações, o objetivo de cada uma era encontrar uma solução que atenda, para um valor $N > 3$, às restrições relacionadas ao problema das N-Rainhas. A escolha feita pela dupla dos algoritmos a serem implementados foi influenciada pelas informações encontradas na bibliografia e nas aulas da disciplina de Introdução à Inteligência Artificial, com a finalidade de propor formas de comparar diferentes abordagens e soluções entre si.

Para alcançarmos os resultados finais utilizamos como base implementações já existentes e informações encontradas nas revisão sistemática do assunto em questão. Houve a documentação das principais características de cada implementação e dos desafios encontrados em cada uma delas.

3.1. Detalhes de Implementação

É importante sinalizar inicialmente que os testes foram realizados em um único *notebook* que portava sistema operacional *Linux* e uma CPU AMD *Ryzen*™ 7 5800H *with Radeon*™ *Graphics* × 16. Lembrando que foi feito uma média referente aos tempos gasto na etapa de testes de algumas implementações, pois dependendo da configuração inicial gerada aleatoriamente o algoritmo pode gastar mais ou menos tempo.

3.1.1. *Backtracking* (Retrocesso)

Na codificação do algoritmo de *Backtracking* foi criada uma função recursiva 'retrocesso' que recebe um inteiro que simboliza a coluna, na qual o algoritmo está tentando encaixar uma rainha.

A estrutura de dados que representa o tabuleiro é uma matriz NxN, contendo as rainhas posicionadas com o valor igual a 1 e o valor 0 para onde não existe uma rainha alocada. O caso base dessa função é quando se é passado uma coluna maior que as dimensões do tabuleiro que envolve o caso.

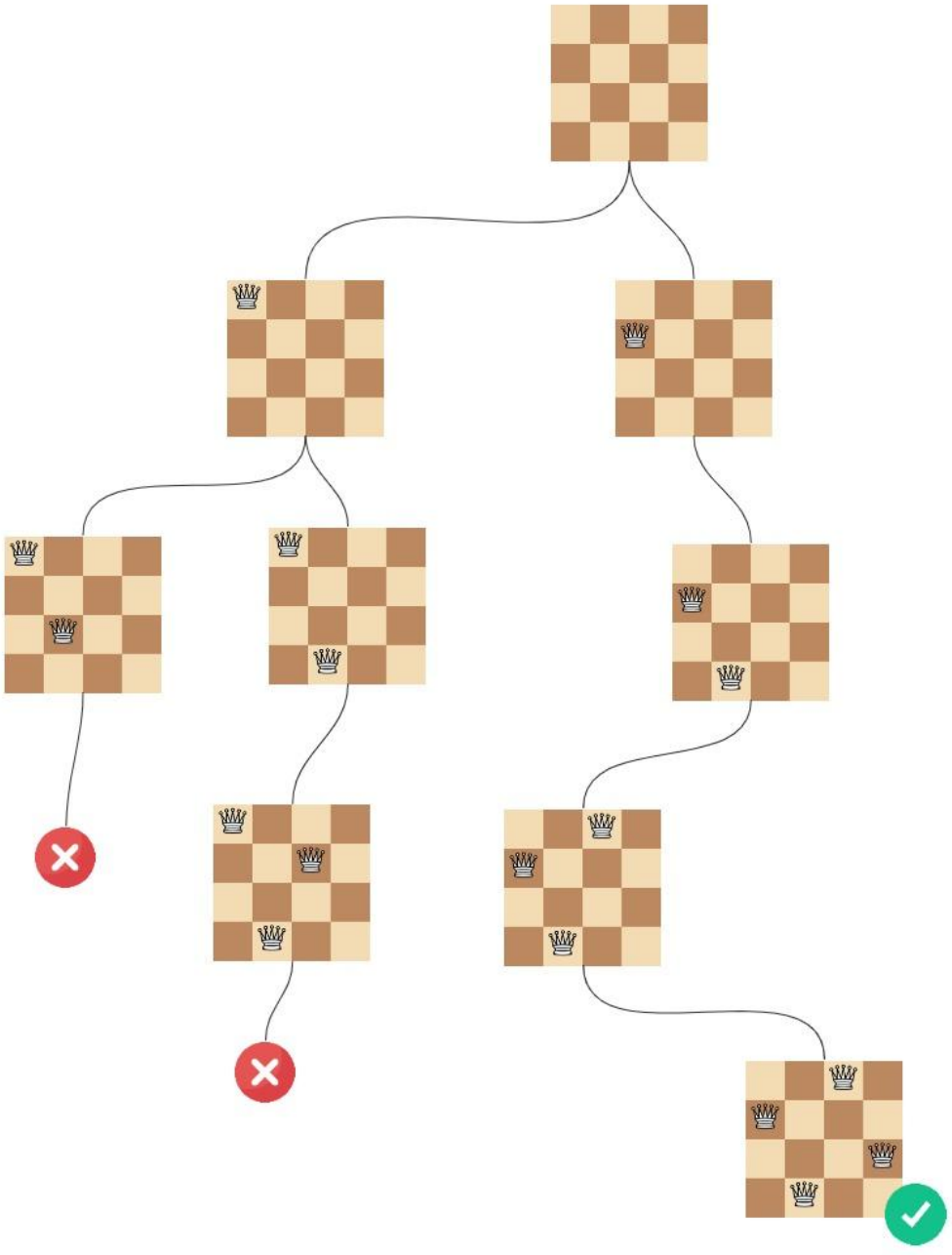
Feito isso, o algoritmo acessa cada linha da coluna e tenta atualizar o *array* do tabuleiro com aquela linha. Ao fazer isso, é realizado cálculos que verificam se uma rainha nessa posição está em ataque, caso a rainha esteja, o tabuleiro não é atualizado e passa para a próxima linha; caso a rainha esteja segura o algoritmo realiza uma chamada recursiva para as próximas colunas, caso não seja possível adicionar uma rainha nas próximas colunas é feito o retrocesso e o algoritmo tentará colocar uma rainha em uma outra linha. Caso tudo ocorra bem e o algoritmo consiga distribuir as rainhas atendendo todas as restrições, a função retornará verdadeiro.

```
função retrocesso(col:inteiro):
        // caso de parada
        if (col ⩾ tamanhoDoBoard)
            retorne verdadeiro;

        para (inteiro i = 0; i < tamanhoDoBoard; i++) faca
            se (adicionarUmaRainha( i, col)) entao
                se(retrocesso(col + 1)) entao
                    retorne verdadeiro;
                  FIM_SE
                removaUmaRainha( i, col); // BACKTRACK
            FIM_SE
        FIM_PARA
```

```
        retorne false;
FIM_FUNÇÃO
```

Figura 1 – Processo do Algoritmo Backtracking



O algoritmo conseguiu chegar em algumas soluções para o problema e, o que foi visto nessas respostas geradas é que, devido a forma como o algoritmo trabalha, as soluções selecionadas em geral possuíam a primeira casa à esquerda com uma rainha, em casos que existiam resoluções com essa característica.

Quanto a performance, o algoritmo acabou crescendo em tempo de um valor N para outro, visto em alguns testes na tabela abaixo. Testes com tempo sinalizados por "-" foram tentados, porém devido a questões de *hardware*, não foram terminados.

Tabela 1 – Casos de Teste do Algoritmo Backtracking e Resultados

***Input*(N)**	**Tempo Gasto(ms)**
8	1
16	19
24	204
32	62101
40	+ de 2 horas
64	-
100	-

A busca por regressão se mostrou interessante para casos em que o valor de N era bem baixo, porém quando esse valor subia o tempo aumentava drasticamente podendo levar até horas sem uma solução ser encontrada por causa do processamento e das incursões envolvidas. O algoritmo também mostrou devolver respostas iguais para as mesmas entradas.

3.1.2. Heurística *Minimum Conflicts*

A heurística de conflitos mínimos expressa uma estratégia diferente do algoritmo de retrocesso inserindo uma lógica para melhorar o estado do tabuleiro durante as suas interações.

Esse algoritmo funciona a partir de uma disposição inicial e aleatória de rainhas no tabuleiro, que usa a mesma estrutura de dados do algoritmo de retrocesso. A cada iteração, o algoritmo seleciona aleatoriamente uma rainha com conflitos e atribui a ela o valor que minimiza o número desses conflitos. Se houver mais de um valor com o mesmo número mínimo de conflitos, um deles é escolhido aleatoriamente.

```
função CONFLITOS-MÍNIMOS(max_etapas: inteiro):
     corrente ← uma atribuição inicial aleatória do problema
     para i = 1 para max_etapas faça
          se corrente é uma solução para n-rainha então
               retornar corrente
          FIM_SE

     var ← uma variável em conflito escolhida aleatoriamente a
     partir das rainhas
     valor ← o valor v para var que minimiza CONFLITOS(var, v,
     corrente, psr)
     definir var = valor em corrente
     FIM_PARA
     retornar falha
FIM_FUNCAO
```

No desenvolvimento desse algoritmo foi realizado a construção de diferentes abordagens para seleção da rainha que seria movida entre as linhas, sendo elas: seleção da rainha com menor conflitos; seleção cíclica de todas as rainhas em ordem da mais a esquerda até a mais a direita; seleção cíclica de todas as rainhas de forma aleatória e seleção de rainhas aleatórias.

A última abordagem foi a que trouxe melhores resultados conseguindo chegar em boas soluções rapidamente. O teste empírico feito nos mostrou que se uma coluna é selecionada muitas vezes seguidas o algoritmo tende a não ser eficiente, mas caso haja uma distribuição igual das vezes em que uma coluna é selecionada, o algoritmo apresentava problemas para convergir para uma solução.

A solução aleatória se mostrou excelente, fazendo com que o algoritmo retornasse em curto período boas soluções. Porém, para essa última estratégia quando tratamos de N muito grande, ou seja > 90, o algoritmo mostrou alguns problemas de conversão para a solução final, e com isso a dupla suspeitou que isso se deve a quantidade maior de colunas a serem sorteadas, onde algumas rainhas em conflitos podem ter sido selecionadas poucas vezes ou até

mesmo nenhuma vez em todo processo do algoritmo. O que foi feito a partir disso foi a reexecução do algoritmo mais de uma vez até que se consiga um resultado.

Graças a inserção de uma estratégia com variáveis aleatórias, nem toda a execução do algoritmo com a mesma entrada N retorna a mesma solução. Quanto à performance, o algoritmo acabou se mostrando eficiente em encontrar solução em um tempo menor quando se comparado ao algoritmo de retrocesso.

Tabela 2 – Casos de Teste da Heurística Minimum Conflicts e Resultados

***Input*(N)**	**Tempo Gasto(ms)**
8	4
16	6
24	8
32	8
40	8
64	18
100	38

3.1.3. *Propagation*

A parte de propagação foi uma maneira encontrada pela dupla de melhorar o desempenho dos algoritmos de retrocesso e de conflitos mínimos. A propagação de restrição foi interligada com essas buscas e consistiu em realizar mudanças nas estruturas de dados que representam os tabuleiros.

Agora, existe uma matriz que registra quantos conflitos cada célula possui, o que evita o excesso de computação que era feito ao fazer a checagem de cada célula. As posições das rainhas ficam em um *array* 0 até N-1 posições a parte contendo a informação das linhas onde elas estão cituadas.

Tabela 3 – Casos de Teste de Propagation com Backtracking e Resultados

***Backtracking* (Retrocesso)**	
***Input*(N)**	**Tempo Gasto(ms)**
8	0
16	6
24	158
32	17661
40	+ de 2 horas
64	-
100	-

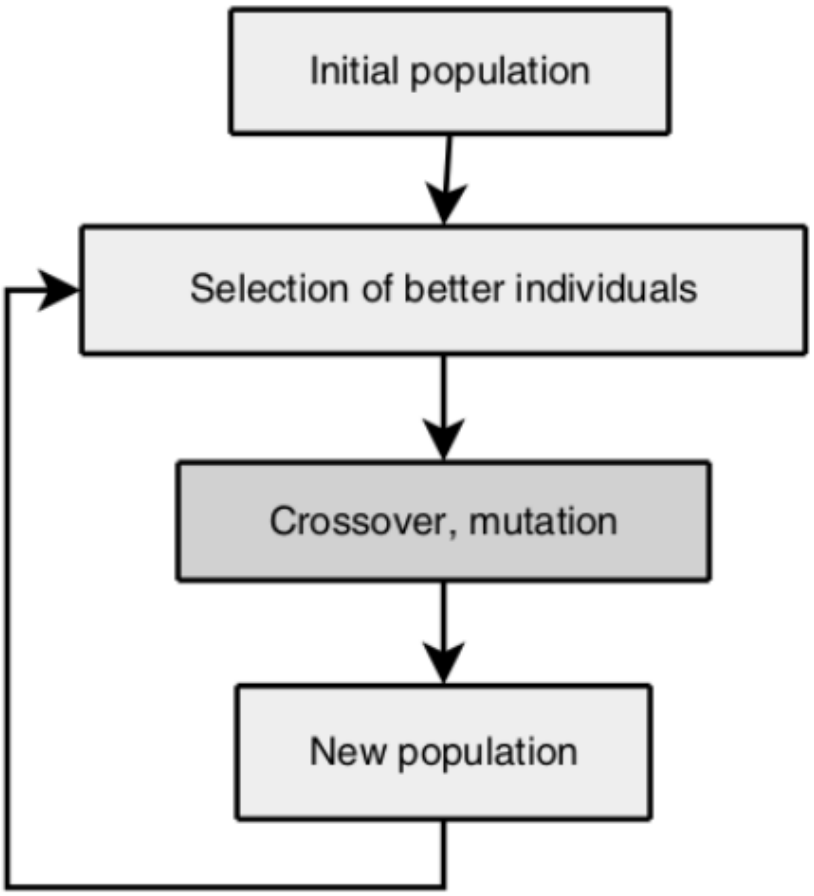
Tabela 4 – Casos de Teste de Propagation com Minimum Conflicts e Resultados

Heurística *Minimum Conflicts*	
***Input*(N)**	**Tempo Gasto(ms)**
8	4
16	5
24	7
32	6
40	6
64	6
100	7

3.1.4. Algoritmo Genético

Foi realizado o desenvolvimento de uma solução baseada em algoritmos genéticos. Esse algoritmo possui a flexibilidade de ser mesclado com outros algoritmos, porém os genéticos devem possuir as seguintes etapas: população inicial, seleção e *crossover*, como visto na imagem abaixo.

Figura 2 – Etapas do Algoritmo Genético



Em um primeiro modelo criado, iniciava-se uma população aleatória de vinte indivíduos, que era representados por um *array*, que continham os genes (posições das rainhas). A função *fitness* criada retornava a soma dos conflitos entre as rainhas, dessa maneira, quanto menor esse valor menos rainhas estariam em conflito, e para solução o algoritmo deveria encontrar um indivíduo com o *fitness* igual a zero.

A parte de *crossover*, onde é realizada o cruzamento sexuado dos indivíduos, foi utilizado um método de roleta no qual depois de escolhido para o cruzamento, o indivíduo era removido das novas seleções, evitando uma convergência prematura dos cromossomos. Os valores de mutação ficaram em torno de 6%, onde uma rainha aleatória era movida de uma

coluna para outra. Essa primeira solução se mostrou desastrosa, não conseguindo convergir para um indivíduo com *fitness* zero nem mesmo em problemas onde o valor de N era pequeno, como 4 ou 8.

A fim de promover uma melhoria nos resultados foi criado uma adaptação da heurística de *Minimum Conflicts* onde as rainhas deveriam ser posicionadas em linhas estratégicas onde se encontrava o menor número de conflitos naquele momento. Essa adaptação foi aplicada primeiramente na população inicial, o que mostrou ser interessante. Para valores de N não muito grandes, era possível conseguir resultados sem a necessidade de muitas iterações no algoritmo, essa adaptação já conseguia resolver alguns casos, mas não necessariamente resolveria todas as entradas sozinha.

Posteriormente, a mesma técnica foi aplicada no algoritmo de mutação. Com essas mudanças, o algoritmo se mostrou eficaz em resolver problemas com valores de N pequenos e, dependendo da população inicial escolhida, resolver casos em que o valor de N > 100, o que diferencia da implementação do *Minimum Conflicts* que tinha certos problemas com valores de N muito grandes.

Foi observado também uma rápida convergência da média de *fitness* de uma população, como visto no gráfico abaixo:

Figura 3 – Convergência Média da Função Fitness

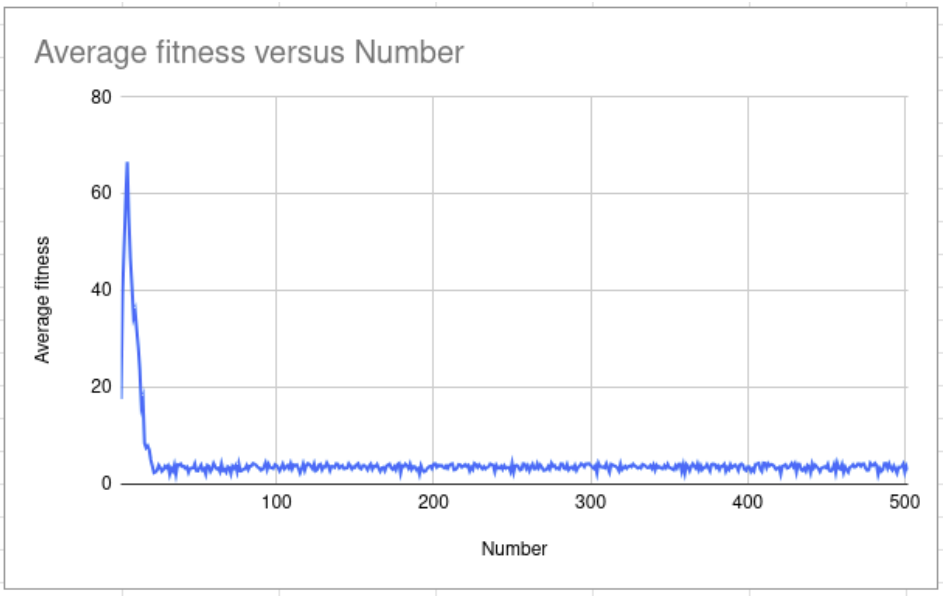


Tabela 5 – Casos de Teste do Algoritmo Genético e Resultados

***Input*(N)**	**Tempo Gasto(ms)**
8	15
16	41
24	44
32	77
40	49
64	90
100	310

Notou-se que em alguns casos há uma convergência das características populacionais, fazendo com que o algoritmo explore uma parte em específico do conjunto universo de soluções, não encontrando, dessa maneira, a solução do problema.

4. ANÁLISE DE RESULTADOS

Essa seção traz um resumo dos casos de testes que foram utilizados para cada um dos algoritmos, a fim de estabelecer um comparativo gráfico das soluções encontradas. Lembrando que algoritmos com abordagens aleatórias são mais difíceis de realizar um comparativo, devido a isso os valores presentes são apenas referentes aos testes feitos.

Tabela 6 – Comparação entre os Casos de Testes Aplicados

	Tempos individuais em milissegundos

Input(N)	*Backtracking*	Heurística *Minimum Conflicts*	*Propagation* + *Backtracking*	*Propagation* + *Minimum Conflicts*	Algoritmo Genético
8	1	4	0	4	15
16	19	6	6	5	41
24	204	8	158	7	44
32	62101	8	17661	6	77
40	7200000	8	7200000	6	49
64	-	18	-	6	90
100	-	38	-	7	310

Figura 4 – Gráfico referente a algoritmos com valores aleatórios

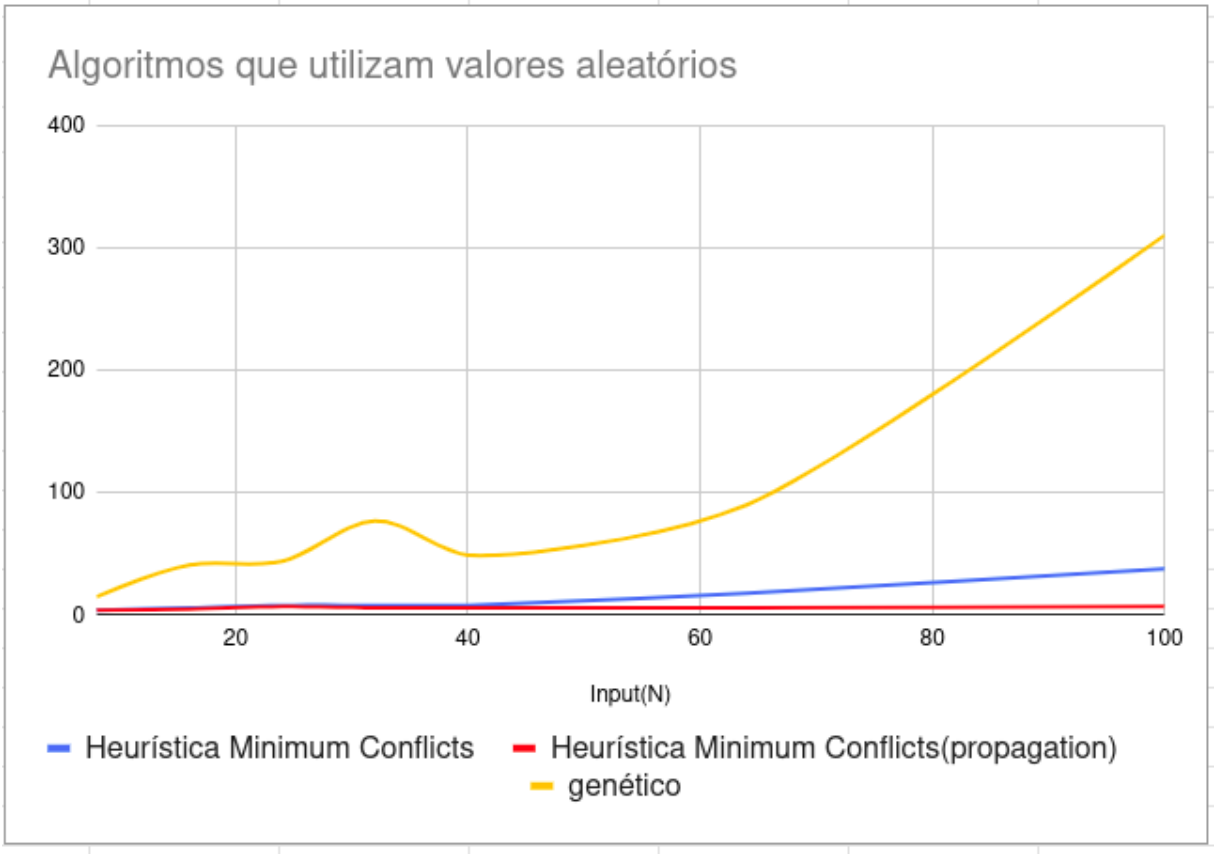
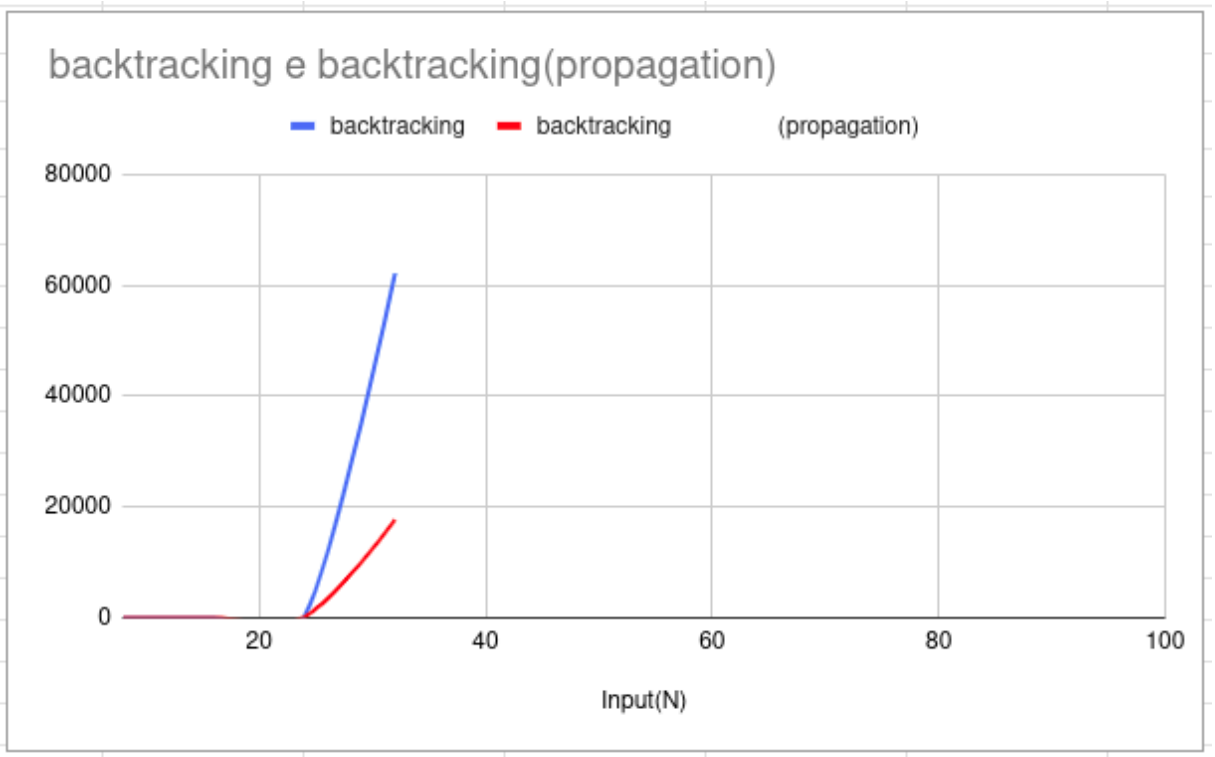


Figura 5 – Gráfico referente a algoritmos sem aleatoriedade



Após os experimentos feitos, conseguimos avaliar o desempenho e analisar os resultados retornados. Com base nas informações expostas, notamos as limitações de cada algoritmo utilizado.

Algoritmos como *Backtracking*, por terem que percorrer de forma sistemática muitas soluções, acabou por ter um tempo de execução muito maior do que se comparado com heurísticas e o Algoritmo Genético. Já o Algoritmo Genético se mostrou interessante resolvendo casos menores com rapidez e podendo resolver casos maiores dadas as condições.

O uso da heurística de conflitos mínimos se mostrou eficiente na resolução de problemas de N-Rainhas e, com o uso de técnicas de Propagação, é possível melhorar ainda mais seu desempenho, porém o algoritmo apresenta problemas com valores de N muito grande, devido ao processo de sorteio de números aleatórios.

5. CONCLUSÃO

Neste relatório, verificamos o conceito do problema das N-Rainhas, praticando de maneira aprofundada determinadas condições de testes após implementação de quatro tipos de algoritmos, tendo assim a oportunidade de visualizar o funcionamento de tais conceitos. No

decorrer do projeto, enfrentamos muitos desafios, principalmente aqueles referentes à performance, e obtemos respostas tanto esperadas, quanto inesperadas. Vamos aprofundar nossas análises.

Sobre os algoritmos, o que retornou melhores valores foi a heurística de *Minimum Conflicts* usando *Propagation*, por causa do seu desempenho e também devido a um custo menor de processamento ocasionado pela técnica de propagação em matriz. Com isso, o projeto demonstrou que é possível implementar diferentes algoritmos para resolver o problema das N-Rainhas, além de ser possível criar algoritmos flexíveis que mesclam mais de uma abordagem na solução.

Em última análise, podemos dizer que a escolha de qual algoritmo utilizar dependerá das necessidades específicas dos parâmetros do usuário, levando em consideração fatores como a complexidade das operações a serem realizadas e do tempo esperado para que esses problemas sejam resolvidos. Cada abordagem apresenta suas particularidades e limitações, por consequência, a seleção da técnica mais apropriada desempenha um papel crucial na eficiência do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

Destrinchando na Elétrica. **Desafio das 8 Rainhas - Teoria e Programação**. *YouTube*, 04 de outubro de 2021. Disponível em: <https://www.youtube.com/watch?v=OzZU9JnK5GY>. Acesso em 27 de dezembro de 2023.

Dias, P. ***Formulations and Exact Algorithms for the Minimum Spanning Tree Problem with Conflicting Edge Pairs***. Belo Horizonte, 2014, 54 f. : il. Dissertação (mestrado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação.

Guilherme, K. **O problema das N rainhas: Solução utilizando Algoritmos Genéticos**. *LinkedIn*, maio de 2023. Disponível em: <https://pt.linkedin.com/pulse/o-problema-das-n-rainhas-solu%C3%A7%C3%A3o-utilizando-gen%C3%A9ticos-kaio-guilherme>. Acesso em 20 de dezembro de 2023.

Minton, S.; Johnston, M.; Philips, A., & Laird, P. ***Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method***. *Eighth National Conference on Artificial Intelligence* (AAAI-90), Boston, Massachusetts: 17–24.

Minton, S.; Johnston, M.; Philips, A., & Laird, P. ***Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems***. *Artificial Intelligence.* 58 (1): 161–205.

Russell, S., & Norvig, P. **Inteligência Artificial - Uma Abordagem Moderna**. 3ª edição. São Paulo: GEN LTC, 2013.

Toffolo, T., & Carvalho, M. ***Backtracking.*** UFOP - Departamento de Computação (DECOM). Disponível em: <http://www3.decom.ufop.br/toffolo/pt-br/ensino/pcc174_2021-1/>. Acesso em 27 de dezembro de 2023.

von Wangenheim, A. ***Backtracking***. UFSC. Disponível em: <https://www.inf.ufsc.br/~aldo.vw/Analise/Backtracking.html>. Acesso em 27 de dezembro de 2023.

Weiss, M.. ***Data Structures & Algorithm Analysis in Java***. Addison-Wesley, 1999.

Witt, T. **O Problema das N-Rainhas**. IME-USP. Disponível em: <https://www.ime.usp.br/~witt/ia/artigo.pdf>. Acesso em 26 de dezembro de 2023.