UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL
***CAMPUS* ARAPIRACA**
CIÊNCIA DA COMPUTAÇÃO

HANDERSON MULLER FERRO DE SOUZA

ALGORITMO EFICIENTE PARA VALIDAÇÃO DE SOLUÇÕES PARA O PROBLEMA DAS N-RAINHAS

ARAPIRACA
2019

Handerson Muller Ferro de Souza

Algoritmo eficiente para validação de soluções para o problema das n-rainhas

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Alagoas - UFAL, Campus de Arapiraca.

Orientador: Prof. Dr. Elthon Allex da Silva Oliveira

Arapiraca
2019

Handerson Muller Ferro De Souza

Algoritmo eficiente para validação de soluções
para o problema das n-rainhas

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Alagoas - UFAL, Campus de Arapiraca.

Data de Aprovação: 30/08/2019.

Banca Examinadora

Prof. Dr. Elthon Allex da Silva Oliveira
Universidade Federal de Alagoas – UFAL
Campus de Arapiraca
Orientador

Prof. Dr. Rodolfo Carneiro Cavalcante
Universidade Federal de Alagoas – UFAL
Campus de Arapiraca
Examinador

Prof. Dr. Tácito Trindade de Araújo Tiburtino Neves
Universidade Federal de Alagoas – UFAL
Campus de Arapiraca
Examinador

Dedico este trabalho a toda minha família que sempre esteve a me apoiar.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus pela presença constante em todos os momentos da minha vida e por sempre iluminar o meu caminho, pois, diante das dificuldades, Ele me conduziu até a conclusão deste curso.

Aos grandes exemplos da minha vida–meus pais José Braz e Gelva Oliveira–, sou grato pela compreensão, incentivo e apoio incondicional, e a minha irmã, por sempre me incentivar e me instigar a perseverar em busca dos meus sonhos de forma digna como também com muito esforço.

A todos os professores do curso de Ciência da Computação que de algum modo tiveram sua parcela no trabalho que apresento hoje e que sempre seguirão comigo em minhas lembranças e nos desafios que terei pela frente.

Aos colegas do curso pelo aprendizado, convivência e amizades conquistadas. Guardo enorme satisfação e carinho por conhecê-los. Em particular, a meu amigo Arthur Max por ser um excelente incentivador e companheiro nos estudos.

Enfim, agradeço a todos aqueles que colaboraram, direta e indiretamente, para que este curso fosse concluído com êxito.

.

“E conhecereis a verdade, e a verdade vos libertará.”

João 8:32

RESUMO

O problema das n-rainhas é um problema clássico da Computação e tomou significativa importância ao ser demonstrado como um problema da classe NP-completo. A partir de sua formulação em 1850, muitas técnicas foram desenvolvidas para sua resolução, desde as mais grosseiras (como força-bruta e *backtracking*) até as mais sofisticadas (como redes neurais e programação por inteiros). O presente trabalho apresenta um novo algoritmo para o problema, observando-o por outra perspectiva: a de validar soluções (ou conjuntos de dados) pré-existentes que podem ter as características de uma solução válida para as n-rainhas.

Palavras-chave: Rainhas. Xadrez. Algoritmo. Validação.

ABSTRACT

The n-queens problem is a classic computer problem and has taken on significant importance as it is demonstrated as an NP-complete class problem. Since its formulation in 1850, many techniques have been developed for its resolution, from the crudest (such as brute-force and backtracking) to the most sophisticated (such as neural networks and integer programming). The present work presents a new algorithm for the problem, looking at it from another perspective: to validate pre-existing solutions (or data sets) that may have the characteristics of a valid solution for n-queens.

Key-words: Queens. Chess. Algorithm. Validation.

LISTA DE FIGURAS

LISTA DE QUADROS E GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

MP	Modelo Padrão
MPM	Modelo Padrão Melhorado
ME	Modelo Eficiente

SUMÁRIO

1 INTRODUÇÃO

A Teoria da Computação é uma área da Computação que tem como objetivo o estudo de questões teóricas da computação com grande relevância, como por exemplo, se um problema seria computável ou não, e se for, se seria possível resolvê-lo em tempo aceitável. Uma subárea dela é o estudo da complexidade computacional, o qual visa determinar a complexidade de problemas e classificá-los conforme sua dificuldade de resolução(ou custo computacional para se obter uma resposta). Dentro desta subárea, seu maior destaque é a questão P vs NP.

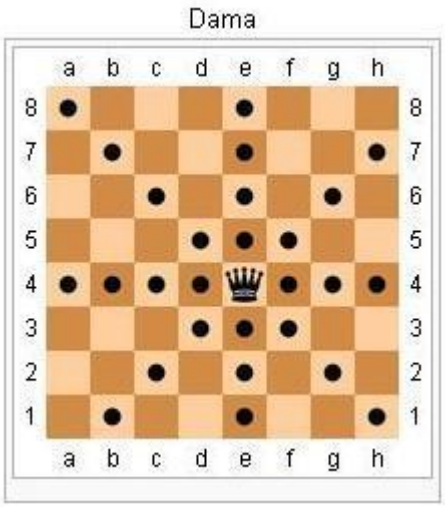
O problema das 8 rainhas foi proposto pela primeira vez por Max Bezzel em 1848[1], o qual tinha como desafio preencher um tabuleiro de xadrez com 8 rainhas. Neste tabuleiro nenhuma delas poderia conflitar com a outra, levando em consideração os movimentos de ataque da peça (como mostrado nas Figuras 1 e 2).

Então, em 1850, o problema foi generalizado por Nauck, que propôs um valor arbitrário n para a quantidade de rainhas dispostas no tabuleiro e criou uma nova variação do problema: a completude das n-rainhas.[38]

No trabalho[40] feito por Gent, Jefferson e Nightingale, foi provado que o problema da completude das n-rainhas faz parte da classe de problemas NP-completo, o que levou as n-rainhas a um novo grau de importância. Ao longo dos anos, diversas técnicas foram desenvolvidas para o problema, como por exemplo, técnicas de busca por *backtracking*[23], métodos heurísticos de busca, busca local e minimização de conflito, redes neurais, programação de inteiros, os quais podem ser observados no artigo de Erbas[20]. Foi proposto também um algoritmo capaz de resolver o problema em tempo $O(n^2)$ através de programação em DNA[31]. Contudo, este algoritmo é inviável para máquinas eletrônicas devido a característica não-determinística que o DNA possui.

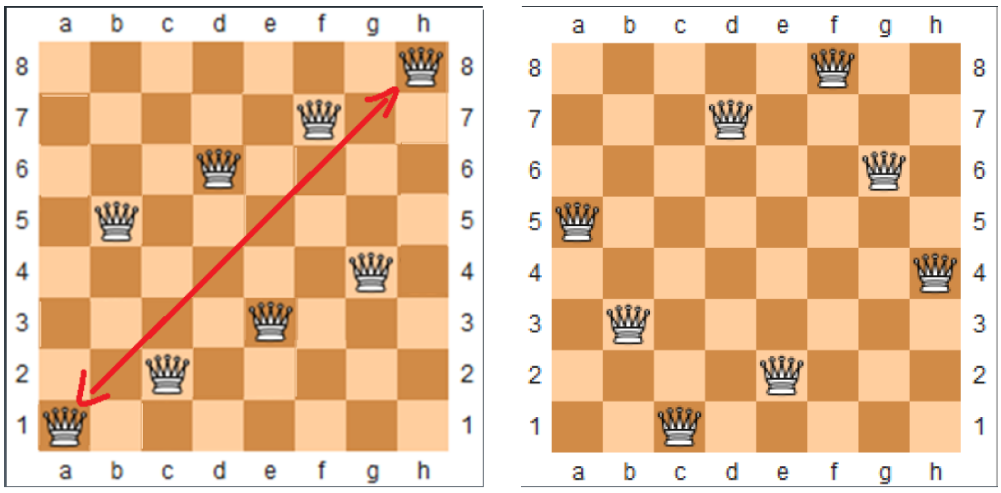
O problema das n-rainhas ainda não possui solução (ótima), assim como todos os NPs, mas este trabalho visa observar o problema por outra perspectiva e trazer um algoritmo que possa ser útil no incremento do estado da arte. Apesar de não haver solução definitiva para o problema, as aplicações práticas para modelagem das n-rainhas são inúmeras, como, prevenção de *deadlock,* teste de VLSI(*Very Large Scale Integration*), controle de tráfego e esquemas de armazenamento para memória paralela, citadas por Erbas[19].

Figura 1 - Movimento de ataque da peça da rainha em um jogo de xadrez.



Fonte: Disponível em: https://segundainfancia.wordpress.com/2013/11/19/jogo-de-xadrez/. Acesso em: 10 abril 2019.

Figura 2 - Uma solução incorreta VS uma solução válida.



Fonte: O autor (2018).

Neste trabalho é apresentado um algoritmo eficiente que tem como intuito validar qualquer conjunto de rainhas proposto como solução para o problema das n-rainhas. Ele foi dividido nos seguintes capítulos: **2. Revisão da literatura**, onde é feita uma visita na história do problema e os diversos algoritmos e técnicas já construídos; **3. Fundamentação teórica**, onde nos é dada à base teórica para compreensão do problema; **4. Construção dos algoritmos e validação**, aqui construímos o algoritmo e demonstramos sua validade; **5. Resultados e trabalhos futuros**, mostramos os resultados obtidos e planos para o futuro.

2 REVISÃO DA LITERATURA

A revisão da literatura teve início utilizando-se a ferramenta de pesquisa do Google com o termo “n-queens problem”, onde foi possível encontrar um repositório[36] sobre o problema. Os trabalhos no repositório datam desde 1848 até 2016, o que o tornou a fonte principal de pesquisa, uma vez que possui a maior parte da literatura sobre o problema. Além deste repositório alguns artigos foram encontrados na pesquisa de periódicos da Capes, com o termo de pesquisa “n-queens”.

Para a execução deste trabalho foram analisados cerca de mais de 100 textos, os quais apenas os mais relevantes serão citados nessa revisão.

2.1 SÍNTESE DOS ARTIGOS

O problema das rainhas começou em 1848[1], quando o enxadrista Max Bezzel propôs o problema das 8 rainhas como um desafio para os leitores da revista e a partir de então muito matemáticos e entusiastas tentaram no resolver, como o conhecido matemático Friedrich Gauss.

Mas apenas em 1850 Franz Nauck apresentou as 92 soluções para o problema e o generalizou, sugerindo uma quantidade N de rainhas para um tabuleiro NxN. Também definiu uma nova instância do problema, o qual foi chamada de “A completude das n-rainhas”. Nessa nova variação do problema eram predefinidas algumas rainhas no tabuleiro o qual se desejava completar para formar uma solução. E a grande pergunta era “Existe solução possível com estas rainhas já colocadas? Se sim, quais são?”.[38]

Em 1889, Sprague apresenta[50] um estudo detalhado sobre o problema das 8 rainhas, demonstrando que para cada solução existente, podem existir mais 7 derivadas da mesma. Ele determinou 3 operações para obtenção dessas soluções derivadas, os quais ele chamou de *Inversion*, *Perversion* e *Reversion*. Também definiu que apesar de existirem 92 soluções para o problema das 8 rainhas, apenas 12 são consideradas distintas.

Neste trabalho ele tentou encontrar alguma regra entre as soluções totais existentes para os tabuleiros de tamanho 4 até 9, o qual não obtive sucesso. Ele percebeu que nem toda solução de um tabuleiro de x rainhas pode gerar uma solução para um tabuleiro de $x + 1$ rainhas.

Em 1901, o livro “*Mathematische Unterhaltungen und Spiele”* escrito por Ahrens[2], apresenta um estudo sobre o problema das n-rainhas, no qual é definido e provado

matematicamente um padrão de construção para soluções. É demonstrado que para dado um tabuleiro de tamanho N maior ou igual a 4, sempre poderá ser construída uma solução válida para o mesmo. Essas soluções futuramente serão conhecidas como “Solução explícita”.

Saltando um pouco no tempo, já em 1965 o problema começou a tomar o interesse não só dos matemáticos, mas de cientistas da computação. Estes começaram a usá-lo como ferramenta de teste para novas técnicas de programação, como a programação *backtracking*, como feito por Golomb[23].

Em 1969, Hoffman apresenta[26] outra demonstração sobre a solução explícita, desta vez mais resumida e de fácil compreensão, o que a fez erroneamente ser citada durante a história como a primeira demonstração da solução explícita. Além de Hoffman outras demonstrações foram feitas antes e depois, apenas mudando alguns aspectos de como se devem ser construídas as soluções.

Em 1977, Campbell faz uma revisão histórica[8] sobre o problema das 8 rainhas e a sua relação com o matemático Gauss, que até então erroneamente vinha sendo colocado como o primeiro a resolvê-lo por completo. Campbell apresenta artigos e cartas da época demonstrando que Nauck nunca desafiou Gauss a resolver o problema e que este apenas o leu em um jornal. Ele alerta sobre como este erro tem sido espalhado durante a história do problema e que muitos livros importantes matemáticos tem feito esta atribuição incorreta.

Já em 1982, Topor apresenta[52] um algoritmo que gera apenas soluções distintas, uma vez que os algoritmos até então tem usado recursos desnecessários, já que para cada solução encontrada, podem existir até mais 7 iguais. Ele define que seu algoritmo é direto e objetivo sobre sua execução, tentando ser o mais otimizado possível.

Em 1984, Foulds aborda[22] o problema por duas perspectivas: grafos e programação de inteiros. Ele demonstra que modelando um tabuleiro de tamanho N para um grafo G e que cada casa do tabuleiro seja um vértice V do grafo, as arestas entre os vértices representam um não ataque, ou seja, dados dois vértices quaisquer existirá uma aresta entre eles se e somente se esses vértices (ou casas) não se ataquem. Deste modo ele prova em seguida que todo subgrafo completo de N vértices de G é uma solução para o problema das n-rainhas em um tabuleiro NxN.

Logo após, ele aborda o problema na perspectiva da programação de inteiros e se prolonga e aprofunda em como construir uma solução válida para o problema diante desta visão.

Em 1986, Clapp apresenta[11] um algoritmo implementado em Ada para resolver o problema das n-rainhas. Ele discute sobre 3 modos diferentes de execução, que são o

sequencial, *Spawn* e *Square*. Os quais ele explica detalhadamente como funcionam no artigo. Com os tempos de execução em uma tabela ele conclui que o sequencial é o mais eficiente, uma vez que não consome nenhum tempo agendando tarefas.

Cockayne e Hedetniemi apresentam[12] uma nota sobre uma variação do problema das n-rainhas: o problema da dominação ou cobertura do tabuleiro. Neste problema é questionado quantas rainhas serão necessárias para que todo o tabuleiro seja coberto por seus campos de ataque, em outras palavras, qual a quantidade mínima de rainhas pra cobrir um tabuleiro NxN. No problema da dominação são admitidos que as rainhas se ataquem entre si. Caso seja definido que elas não possam se atacar, então este será chamado de problema da independência das n-rainhas.

Mas a nota de Cockayne restringe ainda mais esse problema e questiona quantas rainhas são necessárias tendo como restrição que os conjuntos solução sejam formados apenas por rainhas do tipo (x,y) com $x = y$ e x,y representando a linha e coluna respectivamente no tabuleiro.

Também em 1986, Monsky apresenta[34] duas perguntas sobre outra variação do problema. Nesta versão é definida uma peça chamada *superqueen*, a qual se movimenta como uma rainha, mas os ataques diagonais tem a propriedade de refletir no limite do tabuleiro e continuar a atacar através da reflexão. Então em 1989, o próprio Monsky sugere uma solução para suas perguntas[35].

Em 1987, Reichling apresenta[42] um novo modelo de construção de solução explícita, o qual ele dividi em 4 conjuntos.

Também em 1987, Stone faz um estudo[51] empírico sobre o custo computacional do algoritmo de busca *backtracking* na forma lexicográfica para encontrar a primeira solução para o problema das n-rainhas. Ele chega à conclusão que dependendo de alguns parâmetros é possível evitar o custo exponencial dessa tarefa.

Em 1988, Cockayne apresenta[13] um valor limite para a quantidade mínima de rainhas necessárias para resolver o problema da independência das n-rainhas. Ele usa de teoremas e provas matemáticas durante seu artigo até chegar à prova final de seu valor limite.

Em 1989, Abramson[1] apresenta um novo algoritmo para resolver o problema das n-rainhas através da perspectiva de “Divisão e Conquista”, no qual ele dividiu o tabuleiro em sub-tabuleiros. Resolvendo estes tabuleiros menores, ele se utiliza de propriedades matemáticas para garantir que quando construa o tabuleiro maior (a partir da união dos menores) este seja uma solução válida para o problema.

Ele também aborda o problema na sua versão de *superqueens* e mostra que existe uma

relação entre as soluções do problema original e sua variação. Esta solução proposta por Abramson também é um exemplo de solução explícita para o problema.

Já em 1990, Nadel apresenta[37] uma classe de problemas de interesse da IA conhecida como "*Constraint Satisfaction Problem*" ou CSP. Ele modela o problema das n-rainhas nessa perspectiva em diversas representações e compara suas complexidades de resolução tanto empírica quanto teórica.

Cockayne faz um *survey*[14] com os teoremas já construídos sobre o problema da dominação das n-rainhas (o qual já foi abordado por ele[12]), reunindo diversos conteúdos e demonstrando provas sobre esta variação do problema original.

Grinstead[25] também aborda o problema da dominação das n-rainhas, onde ao invés de N ele usa M. Deste modo ele define duas funções $f(m)$ e $g(m)$, onde elas representam respectivamente o número mínimo de rainhas para cobrir o tabuleiro e o número mínimo de rainhas não-atacantes para fazê-lo também. Ele propõe neste trabalho dois limites para essas funções, sendo um para cada e demonstra sua prova.

Já Kalé apresenta[29] uma nova técnica heurística para encontrar a primeira solução de um tabuleiro de tamanho N, o qual ele chamou de "Quase perfeita", pelo fato de ela se demonstrar bem eficiente nos testes com N de 4 até 1000, uma vez que foi reduzido drasticamente a quantidade de *backtrackings* feitos na busca das soluções com essa heurística.

Ainda em 1990, Sosic apresenta[48] um novo algoritmo probabilístico de busca local, baseado em uma heurística gradiente. Ele efetuou testes com N até 500 mil e demonstra seus resultados em tabelas, com observação para algumas variáveis de execução, como, número máximo de rainhas geradas na mesma diagonal por permutação, tempo real de execução do algoritmo, número de colisões geradas por uma permutação e a característica probabilística do algoritmo.

Em 1991, Sosic apresenta[49] uma revisão curta sobre os seus algoritmos construídos para o problema das n-rainhas e mostra uma nova versão o qual ele chamou de QS4. Nesta versão melhorada ele consegue encontrar uma solução para um tabuleiro com 3 milhões de rainhas em menos de um minuto.

Crawford em 1992 apresenta[16] dois algoritmos genéticos para resolver o problema das n-rainhas, o qual ele explica seus funcionamentos e compara seus resultados em seu trabalho.

Já Erbas apresenta[19] um trabalho muito similar a um *survey*, uma vez que este apenas une os diversos algoritmos já produzidos até então sobre o problema das n-rainhas sem nenhum incremento ao estado da arte. Neste trabalho ele demonstra as diversas perspectivas o qual o problema pode ser modelado e dividiu os algoritmos para o problema em 3 categorias:

1- Algoritmos geradores de todas as soluções, 2- Algoritmos geradores de soluções fundamentais e 3- Algoritmos geradores de algumas soluções.

Ainda Erbas apresenta[20] um conjunto de equações de congruência linear para gerar soluções para qualquer N. Suas soluções são similares às soluções explícitas já apresentadas até então, com um diferencial, suas equações permitem gerar mais soluções que apenas uma por N conforme o valor de N cresça.

Homaifar apresenta[27] um algoritmo genético para o problema das n-rainhas o qual ele testou com valores de N até 200. Em seu trabalho são demonstradas tabelas com os resultados obtidos e ele define que se não fosse a limitação de memória do momento, maiores valores de N poderiam ser usados para teste.

Já Mandziuk, apresenta[32] um trabalho sobre uma rede neural de Hopfield desenvolvida para resolver o problema das n-rainhas. Ele cita que ela demonstrou bons resultados e que apenas em 1% dos testes ela caiu em um "mínimo local". Duas estratégias foram empregadas nesse algoritmo e os resultados de sua execução são apresentados.

Ainda em 1992, Rivin apresenta[43] uma solução para o problema das n-rainhas através de um algoritmo de programação dinâmica, o qual ele estipula funcionar com custo computacional $O(f(n)8^n)$, onde $f(n)$ seria uma função polinomial de baixa ordem.

Shagrir apresenta[46] um algoritmo de rede neural Hopfield modificada, onde esta possui a característica de auto inibição em suas unidades. Ele demonstra porque a rede Hopfield padrão é falha para o problema das n-rainhas e mostra que esta modificação que ele fez na rede a faz mais eficiente, provando que ela nunca cai em um "mínimo local".

Em 1993, Gray faz[24] uma pequena revisão sobre o problema das 8-rainhas e apresenta um algoritmo simples para resolver o problema através da perspectiva do *backtracking*. Seu foco neste trabalho não era resolver o problema em si, mas permitir que outros pudessem usá-lo para compreender melhor o problema para que possam construir soluções melhores.

Então em 1994, Cull apresenta[35] o problema das n-rainhas como uma ferramenta para o ensino do conceito de isomorfismo, classes equivalentes e afins. Ele também indica como essas ideias podem ser usadas na prática de programação.

Em 1995, Erbas apresenta[21] uma nova solução explícita para o problema das n-rainhas. Ele se utiliza de matrizes circulantes e se baseia na solução explícita de Hoffman[26], criando super conjuntos de soluções.

Mandziuk apresenta[33] um novo algoritmo de rede neural de Hopfield para resolver o problema. Ele definiu seu modelo como binário, pois a cada instante os neurônios estão em estado 1 ou 0. Ele apresenta tanto um modelo assíncrono, quanto síncrono de funcionamento e

em seus testes demonstra que em ambos os casos houve 100% de convergência. Em sua análise teórica seus dados indicam que o custo computacional médio do algoritmo seja polinomial.

Rolfe apresenta[44] testes de otimização para o algoritmo de *backtracking* das n-rainhas. Em sua metodologia ele aplica testes de execução paralela para o algoritmo. Ele também aborda o problema na perspectiva de como encontrar apenas soluções distintas. Seu trabalho apresenta resultados de execução e as compara em tabelas.

1997, Burguer apresenta[6] novos limites para o número de dominação de um tabuleiro (conhecido como problema da cobertura de um tabuleiro[12]) e para o número de irredundância. Ele se utiliza de provas matemáticas para comprovar estes novos limites em função do valor de N.

Em 1999, Jing apresenta[28] um modelo de vida artificial chamado *ALife*, composto por agentes autônomos distribuídos. Ele aplica este modelo para resolver o problema das n-rainhas e apresenta os seus resultados, demonstrando que o sistema é capaz de resolver o problema para valores grandes de N.

Já em 2000, Silva apresenta[47] um novo algoritmo de rede neural Hopfield modificada. Ele demonstra que a rede é estável e que converge soluções para qualquer valor de N. Ele apresenta seus resultados de teste para validar a proposta de aproximação.

Em 2002, Burger apresenta[7] um novo limite para o número mínimo de rainhas para cobrir um tabuleiro de tamanho N.

Já Zongyan apresenta[55] um algoritmo simples para resolver o problema das n-rainhas através de operações de bit em um array.

Em 2004, Vasquez apresenta dois trabalhos sobre uma variação do problema das n-rainhas: o problema de coloração do grafo de n-rainhas. Nesta versão o desafio é colorir o grafo com a menor quantidade possível de cores de tal maneira que uma rainha não ataque outra de mesma cor. No primeiro artigo[53] ele apresenta algoritmos para esta tarefa, com resultados até N=12 e resultados parciais para N=14. Já no outro artigo[54] ele apresenta novos resultados para esse problema em cima dos algoritmos que vinha desenvolvendo.

2006, Ambrus[3] faz contribuições para o estudo do grafo das rainhas. Um grafo é dito assim se este pode ser modelado para um tabuleiro de xadrez onde cada vértice representa uma rainha e cada aresta nesse grafo representa uma rainha atacando a outra, na vertical, horizontal ou diagonal. Em seu trabalho ele apresenta provas sobre conjecturas desse problema e aborda algumas questões em aberto.

Cournia apresenta[15] um algoritmo para computar o menor número de dominação de

um tabuleiro (o problema da cobertura já citado[12]). Seu diferencial é que este foi projetado para trabalhar na GPU e os resultados obtidos mostraram que foi bem mais sucedido que sua versão para CPU, para valores de N maior que 9.

Noguchi apresenta[39] uma rede neural baseada em Hopfield, mas com alterações na maneira como são processadas as conexões na rede. Ele apresenta três modelos próprios e compara-os com outros trabalhos. Ele valida seus modelos através dos bons resultados mostrados.

2009, Bell apresenta um *survey*[4] reunindo os teoremas e conjecturas sobre o problema original das n-rainhas e suas variações. Ele faz um estudo detalhado sobre a história do problema, que vai desde os modelos formais, até as provas matemáticas ao longo dos anos.

Ainda no ano de 2009, Chatham apresenta[9] um estudo sobre o desempenho de algoritmos para uma variação do problema das n-rainhas: o problema das N + k rainhas. Nesta versão, o desafio é colocar N + k rainhas em um tabuleiro de tamanho N, sendo que é permitido o uso de k peões para evitar o ataque de rainhas. Os algoritmos testados em seu trabalho tentam computar todas as soluções possíveis para este problema.

Em 2010, Draa apresenta[18] um novo algoritmo para o problema das n-rainhas baseado nos princípios da computação quântica e dos algoritmos evolucionários. Este híbrido apresentou bons resultados em seus testes.

Já em 2011, San Segundo apresenta[45] em um novo conjunto de regras heurísticas para o algoritmo de *backtracking*, tendo como intuito reduzir o custo de processamento para encontrar as K primeiras soluções para o problema das n-rainhas através da busca por tentativa e erro.

Em 2012, Chatham aborda novamente o problema das N + k rainhas, mas desta vez observando apenas as soluções que possuem alguma característica de simetria. Ele demonstra provas sobre proposições a esse tipo de solução, restringindo os valores de N e k aonde podem ocorrer determinados tipos de simetria.[10]

2013, Maazallahi apresenta[31] um algoritmo para resolver o problema das n-rainhas através da programação em DNA. O grande diferencial deste trabalho é que através desta técnica foi possível resolver o problema com custo O(n^2). Infelizmente ainda não é possível replicar as propriedades de funcionamento do DNA para os computadores feitos de silício, o que torna inviável o uso desta técnica para uso cotidiano.

Em 2016, Preuber apresenta[41] um estudo e modelo de execução paralela para a contagem da quantidade de soluções para o problema das n-rainhas com N=27. Ele utiliza conceitos de simetria e grupos nessa primeira parte para tornar ainda mais efetivo à execução

do projeto. São apresentados também os resultados de execução e *bechmark* dessa tarefa e estimativas do custo de conclusão desse processo são mensurados.

E finalmente em 2017, Gent, Jefferson e Nightingale apresentam[40] um estudo sobre a classe de complexidade do problema da completude das n-rainhas. Eles demonstram que o problema é tanto NP-completo, quanto #P-completo. Sua demonstração é feita através de diversas transformações polinomiais reduzindo o problema original a um problema NP-completo conhecido. Este trabalho foi de grande importância uma vez que foi provado que o problema das rainhas faz parte da classe dos NPs-completo, e portanto, possui todas as características dos problemas dessa classe.

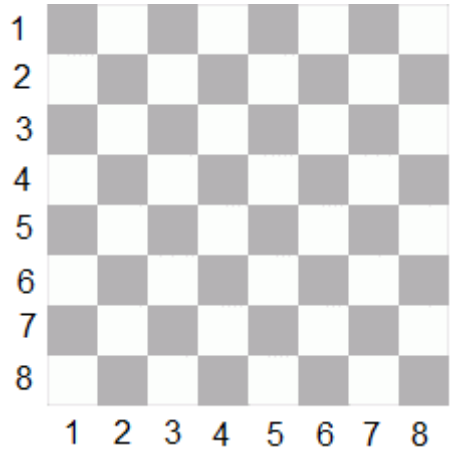
3 FUNDAMENTAÇÃO TEÓRICA

Para que haja compreensão de como os algoritmos do próximo capítulo foram construídos e de que modo suas validações foram feitas, é necessário entender o modelo matemático utilizado neste trabalho para o problema das n-rainhas e o conceito de análise assintótica, usado para validar os custos computacionais dos algoritmos desenvolvidos. Portanto, as subseções a seguir são destinadas a este fim.

3.1 MODELO MATEMÁTICO PARA O PROBLEMA DAS N-RAINHAS

Originalmente o problema das n-rainhas é demonstrado em um tabuleiro de xadrez, então uma maneira simples de moldá-lo é atribuir números para cada linha e coluna no tabuleiro, deste modo, a enumeração das linhas e colunas de 1 até N é o mais plausível a se fazer. Podemos ver um exemplo para N=8 na figura 3.

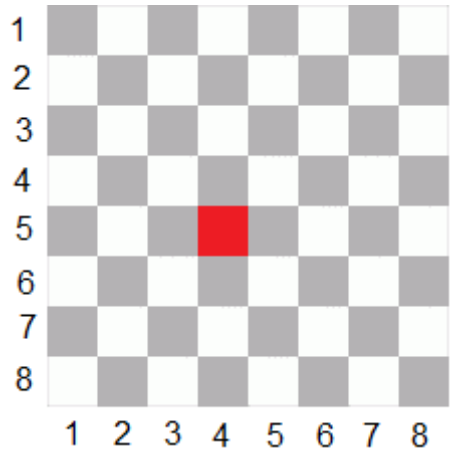
Figura 3 - Tabuleiro de xadrez enumerado com N=8.



Fonte: O autor (2018).

Com o tabuleiro enumerado, podemos representar qualquer posição através de uma tupla, composta pelo número da linha e coluna da casa desejada. Por exemplo: (5,4) seria a representação matemática da posição mostrada na figura 4.

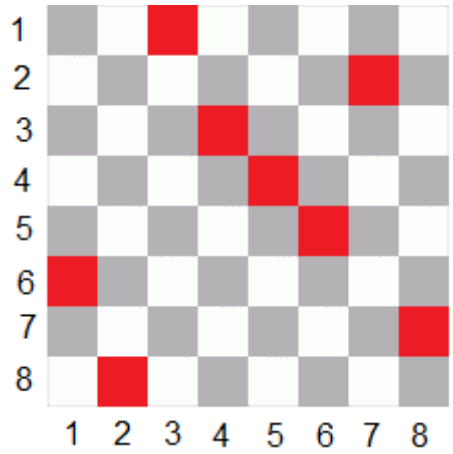
Figura 4 - Representação da posição (5,4) no tabuleiro de xadrez.



Fonte: O autor (2018).

Definido como representar as casas do tabuleiro, podemos usar a mesma modelagem para representar conjuntos de rainhas, que podem ser ou não soluções para uma instância do problema das n-rainhas. Conjuntos com N tuplas seriam representações de N rainhas em um tabuleiro, como por exemplo: {(1,3), (2,7), (3,4), (4,5), (5,6), (6,1), (7,8), (8,2)} é uma representação para um conjunto com N=8 e pode ser visto na figura 5.

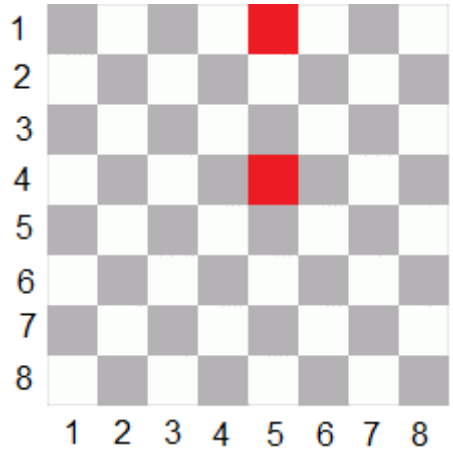
Figura 5 - Um conjunto com 8 rainhas no tabuleiro.



Fonte: O autor (2018).

Por definição, o problema das n-rainhas restringe que nenhuma rainha poderá atacar outra, o que nos dá 3 restrições: sem rainhas na mesma linha, coluna ou diagonal. Com a modelagem usada acima é simples identificar as duas primeiras restrições, pois, rainhas na mesma linha ou coluna terão valores iguais nas respectivas posições em suas tuplas. Por exemplo, as rainhas (4,5) e (1,5) estão na mesma coluna, uma vez que os valores que representam suas respectivas colunas são iguais. Isto pode ser visualizado na figura 6.

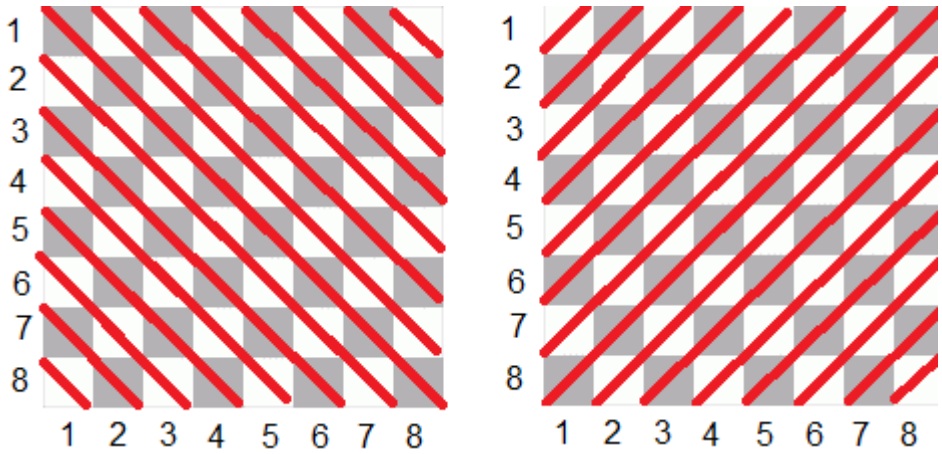
Figura 6 - Rainhas (4,5) e (1,5) estão na mesma coluna.



Fonte: O autor (2018).

Mesmo não sendo tão visível, é possível usar essa modelagem para identificar a 3º restrição, mas o primeiro passo é assumir que existem duas diagonais: a principal e a secundária. Deste modo, cada posição do tabuleiro possui 4 coordenadas: linha, coluna, diagonal principal e diagonal secundária. Uma representação visual dessas diagonais pode ser observada na figura 7, onde o tabuleiro da esquerda representa as diagonais principais, e o tabuleiro da direita as diagonais secundárias.

Figura 7- Diagonais principais (tabuleiro da esquerda) e diagonais secundárias (tabuleiro da direita).



Fonte: O autor (2018).

Para se obter o valor da diagonal principal de uma casa do tabuleiro, somente precisamos subtrair sua linha de sua coluna, deste modo temos que:

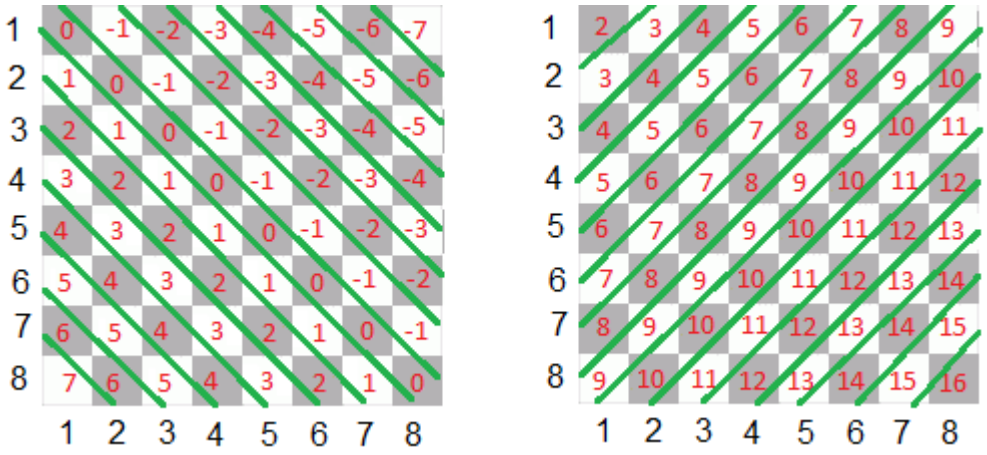
$$linha - coluna = diagonal\ principal$$

E para a diagonal secundária devemos fazer sua soma:

$$linha + coluna = diagonal\ secundária$$

Se aplicarmos essas fórmulas em um tabuleiro de tamanho N=8, poderemos ver as diagonais sendo representadas por um único valor, como na figura 8.

Figura 8 - Representação das diagonais primárias e secundárias através de números.



Fonte: O autor (2018).

Assim como a representação da tupla (linha, coluna) nos permite facilmente identificar quando duas rainhas estão na mesma linha ou coluna, podemos também identificar através dos números gerados quando elas estão na mesma diagonal, uma vez que os mesmos são representações numéricas da diagonal em que as rainhas se encontram. Por exemplo, colocando as rainhas nas posições (5,4) e (2,7) podemos afirmar que as mesmas não estão na mesma linha ou coluna. Aplicando as fórmulas de diagonal obtemos que elas não estão na diagonal principal, mas que estão na diagonal secundária:

Diagonal principal:

$$5 - 4 \neq 2 - 7$$

$$1 \neq -5$$

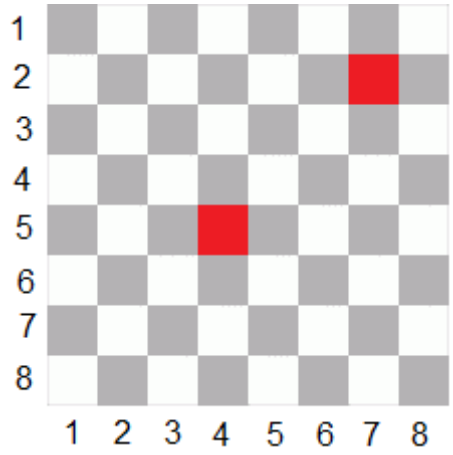
Diagonal secundária:

$$5 + 4 = 2 + 7$$

$$9 = 9$$

Este fato pode ser visto na figura 9 abaixo.

Figura 9 – Rainhas nas posições (5,4) e (2,7) na mesma diagonal secundária.



Fonte: O autor (2018).

Definida a representação das rainhas e como identificar através desta modelagem quando duas rainhas estão em conflito, podemos prosseguir para seção seguinte.

3.2 CONCEITOS DE ANÁLISE ASSINTÓTICA

3.2.1 O que é a análise assintótica ?

Análise assintótica é um método de descrever o comportamento de limites, sejam esses de funções ou algoritmos. Este tipo de análise se preocupa com valores de entradas muito grandes para suas funções e busca definir níveis para funções equivalentes, a fim de normalizar e facilitar a equiparação de custos computacionais entre soluções. Através desta técnica é possível identificar situações as quais algoritmos poderiam entrar em um processo de execução quase infinito, por causa das limitações computacionais atuais.

3.2.2 Notação Assintótica

A análise assintótica dividiu o estudo do custo computacional através da classificação de ordens, onde as mais relevantes são as ordens *Omega*, *Theta*, *O* (conhecido como “O grande”). *Omega* visa analisar qual o menor valor de entrada para uma função, *Theta* analisa o valor que chamamos de “caso médio”, valores que tem certa probabilidade de ocorrer com mais frequência e por fim, *O*, o qual analisa o “pior caso”, o maior valor que poderá entrar e gerar o maior custo computacional. Suas representações podem ser vistas a seguir:

$O(f)$ = O grande da função f

$\Omega(f)$ = Omega da função f

$\Theta(f)$ = Theta da função f

Agora tomemos como exemplo um pseudo-algoritmo no qual é feita a soma de todos o números naturais de 1 até N:

```
N = {conjunto com os N primeiros números naturais}
soma = 0
para cada x em N:
        soma = soma + x

mostrar(soma)
```

Se N for igual a 4 (quatro), teremos 4 iterações no laço de repetição, logo podemos assumir que teremos custo 4 nesta execução. Se N for igual a 10 (dez), custo 10 e assim sucessivamente, portanto concluímos que o custo computacional deste algoritmo está intrinsicamente ligado ao valor de N e que, portanto, a definição de seu valor para ordem O() é N, logo esta função é O(N). Para Θ() temos mais uma vez o valor de N, uma vez que não à possibilidade de variação na execução por causa da ligação com N. E por fim o custo Ω() da função é $\Omega(1)$ uma vez que se N for igual a 0 (zero) apenas a execução do "mostrar(soma)" será executado.

Para mais detalhes sobre análise assintótica e notação assintótica uma leitura complementar do livro de Knuth em suas subseções "1.2.10 - Analysis of an Algorithm" e "1.2.11 - Asymptotic Representations" seria interessante.[30]

Neste trabalho apenas focaremos no uso da notação O() para medir e comparar os algoritmos construídos.

4 CONSTRUÇÃO DOS ALGORITMOS E VALIDAÇÃO

Este capítulo é dedicado ao desenvolvimento dos algoritmos e análise de seus custos computacionais, como também a comparação entre eles através da análise assintótica. Foram produzidos três algoritmos os quais foram chamados de: Modelo padrão (MP), Modelo padrão melhorado (MPM) e Modelo eficiente (ME).

4.1 MODELO PADRÃO

O modelo padrão foi o primeiro algoritmo a ser produzido nesse trabalho e reflete a simplicidade do pensamento comum das pessoas sobre o problema das n-rainhas, como será explicado a seguir.

4.1.1 Imaginando o MP

Sabendo-se das restrições do problema das n-rainhas e de posse do modelo matemático usado para modelá-lo anteriormente, qual o primeiro pensamento que se tem para se verificar se num conjunto de n-rainhas existe alguma rainha que ataque outra ? A resposta é simples e óbvia: compare as rainhas entre si e se nenhuma delas houver uma linha, coluna ou diagonal similar significa que este conjunto de rainhas será uma solução válida, caso contrário não o será.

4.1.2 Construindo o MP

De posse da resposta da subseção anterior precisamos transformá-la em algoritmo e para isso necessitamos raciocinar de que maneira aquelas palavras poderão funcionar. A maneira mais simples que é observável, seria o uso de uma iteração que percorre todas as rainhas de um conjunto, na qual esta iteração estaria dentro de outra iteração que faz o mesmo processo, deste modo poderíamos comparar todas as rainhas entre si, usando os loops da iteração. Este algoritmo pode ser visto abaixo:

conjunto_de_rainhas = {}
verdade = 1 //Variável usada para identificar ao final das iterações se é ou não solução

para cada *rainhaX* em *conjunto_de_rainhas*:

diagonalPriX = *rainhaX.linha* - *rainhaX.coluna*
diagonalSecX = *rainhaX.linha* + *rainhaX.coluna*

para cada *rainhaY* em *conjunto_de_rainhas*:

diagonalPriY = *rainhaY.linha* - *rainhaY.coluna*
diagonalSecY = *rainhaY.linha* + *rainhaY.coluna*

se *rainhaX.linha* IGUAL *rainhaY.linha*:
verdade = 0
se não se *rainhaX.coluna* IGUAL *rainhaY.coluna*:
verdade = 0
se não se *diagonalPriX* IGUAL *diagonalPriY*:
verdade = 0
se não se *diagonalSecX* IGUAL *diagonalSecY*:
verdade = 0

se *verdade* IGUAL 1:
mostrar(“Este conjunto é uma solução válida”)
se não:
mostrar(“Este conjunto não é uma solução válida”)

Apesar do algoritmo acima aparentar estar correto, ele possui uma falha crucial em sua execução, pois no momento inicial as variáveis *rainhaX* e *rainhaY* serão as mesmas, o que fará com que ele sempre retorne que o conjunto não é uma solução válida. Isto pode ser facilmente resolvido colocando-se uma condicional antes do processo de comparação, como visto a seguir:

conjunto_de_rainhas = {}
verdade = 1 //Variável usada para identificar ao final das iterações se é ou não solução

para cada *rainhaX* em *conjunto_de_rainhas*:
diagonalPriX = *rainhaX.linha* - *rainhaX.coluna*
diagonalSecX = *rainhaX.linha* + *rainhaX.coluna*

```
        para cada rainhaY em conjunto_de_rainhas:

            se rainhaX DIFERENTE rainhaY:
                diagonalPriY = rainhaY.linha - rainhaY.coluna
                diagonalSecY = rainhaY.linha + rainhaY.coluna

                se rainhaX.linha IGUAL rainhaY.linha:
                    verdade = 0
                se não se rainhaX.coluna IGUAL rainhaY.coluna:
                    verdade = 0
                se não se diagonalPriX IGUAL diagonalPriY:
                    verdade = 0
                se não se diagonalSecX IGUAL diagonalSecY:
                    verdade = 0

    se verdade IGUAL 1:
        mostrar("Este conjunto é uma solução válida")
    se não:
        mostrar("Este conjunto não é uma solução válida")
```

Agora sim possuímos um MP completo e funcional, uma vez que ele comparará as rainhas entre si evitando a comparação com peças iguais.

4.1.3 Analisando o custo computacional do MP

Normalmente o custo computacional de um algoritmo é avaliado pelo seu fluxo de execução, que neste caso é de cima para baixo. Suas primeiras linhas de execução tem custo O(1), uma vez que executam tarefas simples, por esse motivo iremos ignorá-las no cálculo do custo computacional do MP e quaisquer outras linhas que tenham custo constante irrisório. Nos sobra portanto as duas iterações:

```
        ⋮⋮⋮
    para cada rainhaX em conjunto_de_rainhas:
        ⋮⋮⋮
        para cada rainhaY em conjunto_de_rainhas:
```

⋮⋮⋮

Ambas as iterações estão intrinsecamente ligadas à quantidade de rainhas que possui o conjunto, logo para N rainhas, haverá N iterações. Como uma das iterações está dentro da outra significa que para cada loop feito na exterior, N loops serão feitos na interior, portanto para M loops na exterior, haverá um total de M x N loops na interior. Já que o máximo de iterações para a exterior é N podemos concluir que serão feitos um total de n^2 iterações, consequentemente concluímos que o custo máximo computacional é O(n^2).

4.2 MODELO PADRÃO MELHORADO

De posse do algoritmo MP, nos seria interessante melhorá-lo, uma vez que apesar do custo O(n^2) não ser considerado muito alto, ainda sim não é o mais eficiente.

4.2.1 Imaginando o MPM

Observando a execução do MP e o modo como funciona é possível identificar que sua ineficiência ocorre a partir do momento em que para cada rainha que ele verifica na iteração externa ele a reverifica mais N vezes na iteração interna, uma execução desnecessária já que uma rainha somente precisa ser comparada com todas as outras uma única vez. Se pudermos evitar esta reverificação poderemos reduzir significativamente o custo computacional do MP.

4.2.2 Construindo o MPM

Se pudermos restringir de alguma maneira a quantidade de vezes que uma rainha é verificada, poderemos fazer com que o algoritmo seja mais eficiente e o modo simples de se conseguir isso seria remover as rainhas que já foram verificadas antes. Para isso deveremos observar as posições das rainhas no conjunto de rainhas e usar os índices do conjunto de rainhas para conseguir facilmente essa “remoção”. Podemos ver o algoritmo abaixo para melhor compreensão:

conjunto_de_rainhas = {}
verdade = 1 //Variável usada para identificar ao final das iterações se é ou não solução
n = //Número inteiro que representa a quantidade de rainhas

```
para cada x em {1 até n}:
    rainhaX = conjunto_de_rainhas[x]
    diagonalPriX = rainhaX.linha - rainhaX.coluna
    diagonalSecX = rainhaX.linha + rainhaX.coluna

    para cada y em {x+1 até n}:
        rainhaY = conjunto_de_rainhas[y]
        diagonalPriY = rainhaY.linha - rainhaY.coluna
        diagonalSecY = rainhaY.linha + rainhaY.coluna

        se rainhaX.linha IGUAL rainhaY.linha:
            verdade = 0
        se não se rainhaX.coluna IGUAL rainhaY.coluna:
            verdade = 0
        se não se diagonalPriX IGUAL diagonalPriY:
            verdade = 0
        se não se diagonalSecX IGUAL diagonalSecY:
            verdade = 0

se verdade IGUAL 1:
    mostrar("Este conjunto é uma solução válida")
se não:
    mostrar("Este conjunto não é uma solução válida")
```

A iteração externa continuou selecionando todas as rainhas, mas a interna não. A partir do momento que usamos o artificio "*x*+1" na iteração interna evitamos o problema da verificação de uma rainha com ela mesma, uma vez que a rainha selecionada na iteração interior sempre será a próxima do conjunto. Mas este artificio foi muito mais além, já que através dele foi possível "remover" da comparação todas as rainhas que já tenham sido verificadas antes, uma vez que "criamos" um subconjunto de rainhas na iteração interna através da restrição "{*x*+1 até *n*}".

4.2.3 Analisando o custo computacional do MPM

Mais uma vez devemos ignorar todos os custos irrisórios do algoritmo, o que nos reduz as duas estruturas de loop:

⋮⋮⋮

para cada *x* em {1 até *n*}:

⋮⋮⋮

para cada *y* em {*x*+1 até *n*}:

⋮⋮⋮

É fácil identificar que a primeira estrutura nos dará custo O(n) uma vez que produz n iterações. A segunda estrutura é um pouco mais complicada, mas sabemos que seu custo é inferior a n. Se observamos atentamente perceberemos que a quantidade de iterações feitas na segunda estrutura está intrinsecamente ligada ao valor de x. Quanto maior for o valor de x menor será a quantidade de execuções. Portanto, podemos induzir que a quantidade de iterações secundárias será expressa pelo valor $n - x$.

Sabendo que n é um valor constante e que x é um valor crescente a partir de 1, podemos admitir que o custo computacional de cada loop da iteração interior será dado pela sequência $n-1, n-2, n-3, n-4$ ... . Como a interação exterior tem n loops, podemos admitir que a sequência citada terminará com $n - n$.

Para se medir o custo computacional do MPM devemos somar esta sequência. Se bem observado será possível perceber que a sequência em questão tem um padrão: uma ordem numérica decrescente de números naturais. Como a soma possui a propriedade comutativa, podemos inverter a ordem da soma e dizer que é uma soma dos $n - 1$ primeiros números naturais e assim deste modo usar a fórmula da soma dos n primeiros números naturais:

$$((n + 1) * n)/2$$

No nosso caso substituiremos n na fórmula por $n - 1$, ficando assim:

$$((n - 1 + 1) * (n - 1))/2$$

Simplificando a equação, obtemos:

$$(n^2 - n)/2$$

Portanto o custo máximo computacional do MPM é O($(n^2 - n)/2$).
Onde O($(n^2 - n)/2$) < O(n^2).

4.3 MODELO EFICIENTE

Apesar de o MPM ser mais eficiente que o MP, ambos estão na mesma classe de equivalência de funções, pois ambos possuem n^2como seu grau mais alto. Este fato faz com que teoricamente ambos tenham o mesmo custo de execução. Por esse motivo uma nova abordagem foi necessária ser feita e então surgiu o Modelo eficiente (ME).

4.3.1 Imaginando o ME

Até o momento os MP e MPM usaram uma abordagem onde as rainhas eram escolhidas uma por vez e apenas um pequeno artifício fora utilizado para "remover" rainhas já avaliadas, mas mesmo assim não havia um controle real sobre quantas e quais rainhas tinham sido analisadas. Então partindo do pressuposto de que pudéssemos "armazenar" as rainhas previamente analisadas, talvez fosse possível obter uma redução significativa.

Aprofundando um pouco mais o raciocínio percebesse que o que realmente importa não são as rainhas em si, mas os números que representam suas coordenadas, uma vez que, estes são os responsáveis por identificar conflitos.

Agora que obtemos a capacidade de armazenar, seria uma opção interessante sinalizar a ocorrência das linhas, colunas e diagonais de cada rainha observada, pois deste modo, para cada rainha verificada, seria armazenado a ocorrência de determinada linha, coluna e diagonais. Quando a próxima rainha fosse verificada, apenas observaríamos se alguma de suas coordenadas coincidiria com alguma coordenada que já havia sido previamente armazenada. Deste modo, seria muito mais eficiente a identificação de conflitos, uma vez que cada rainha só precisaria ser analisada uma única vez.

4.3.2 Construindo o ME

Para construirmos um algoritmo que tenha a capacidade de armazenar as coordenadas de cada rainha, só precisamos criar 4 conjuntos vazios, nos quais cada um representará res-

pectivamente uma coordenada: linhas, colunas, diagonais primárias, diagonais secundárias. Este algoritmo poderá ser visto abaixo:

```
conjunto_de_rainhas = {}
verdade = 1 //Variável usada para identificar ao final das iterações se é ou não solução

linhas = {}
colunas = {}
diagonaisP = {} //Para diagonais primárias
diagonaisS = {} //Para diagonais secundárias

para cada rainhaX em conjunto_de_rainhas:
    diagonalPriX = rainhaX.linha - rainhaX.coluna
    diagonalSecX = rainhaX.linha + rainhaX.coluna

    se rainhaX.linha NÃO ESTA EM linhas:
        linhas.colocar(rainhaX.linha)
    se não:
        verdade = 0

    se rainhaX.coluna NÃO ESTA EM colunas:
        colunas.colocar(rainhaX.coluna)
    se não:
        verdade = 0

    se diagonalPriX NÃO ESTA EM diagonaisP:
        diagonaisP.colocar(diagonalPriX)
    se não:
        verdade = 0

    se diagonalSecX NÃO ESTA EM diagonaisS:
        diagonaisS.colocar(diagonalSecX)
    se não:
        verdade = 0
```

se *verdade* IGUAL 1:

    mostrar(“Este conjunto é uma solução válida”)

se não:

    mostrar(“Este conjunto não é uma solução válida”)

O algoritmo acima verifica através de cada condicional se já houve ou não a ocorrência de uma coordenada em seu respectivo conjunto, e se já houve sinaliza através da variável *“verdade”*.

4.3.3 Analisando o custo computacional do ME

Devemos remover do algoritmo as execuções de custo simples e analisar o que restou, deste modo o que nos sobra é sua única iteração:

⋮⋮⋮

para cada *rainhaX* em *conjunto_de_rainhas*:

⋮⋮⋮

Como esta iteração depende da quantidade de rainhas dentro do conjunto é fácil notar que serão executados n loops, portanto, podemos determinar que seu custo computacional é O(n). Um custo muito inferior em comparação com os algoritmos MP e MPM que são de uma classe superior – O(n^2).

5 RESULTADOS E TRABALHOS FUTUROS

Neste último capítulo iremos debater sobre os resultados obtidos através da comparação dos algoritmos em instâncias teóricas de execução e definir metas futuras para a sequência desta pesquisa.

5.1 COMPARAÇÃO DE EXECUÇÃO DOS ALGORITMOS PARA VALORES DE N

Se observarmos o quadro a seguir poderemos perceber as diferenças que existem no custo computacional entre os 3 algoritmos ao atribuirmos valores de N para suas execuções:

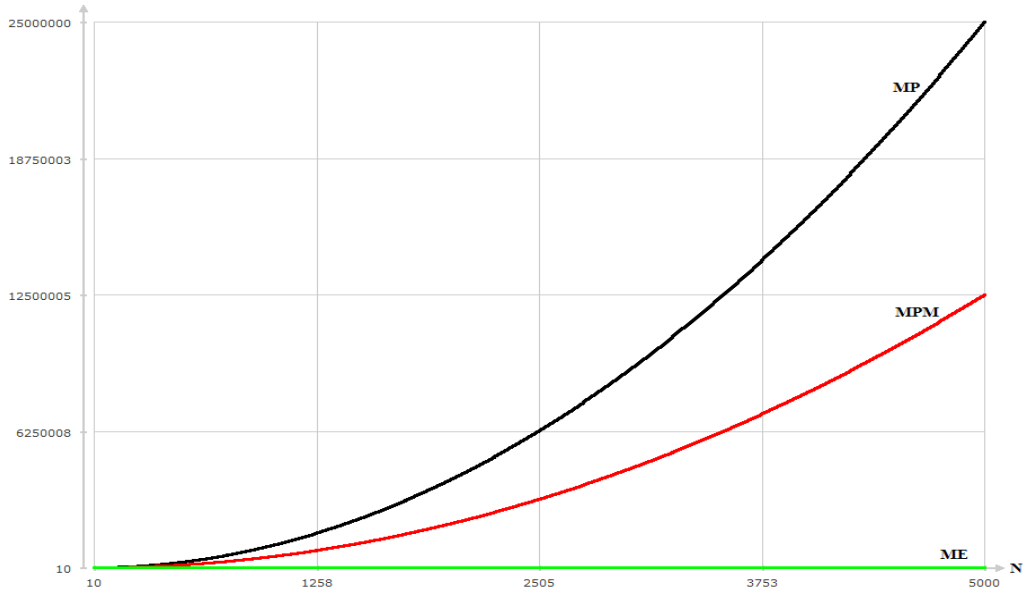
Quadro 1 – Quadro dos custos computacionais com valores de N até 5000.

N=	10	100	1000	5000
MP	100	10000	1000000	25000000
MPM	45	4950	499500	12497500
ME	10	100	1000	5000

Fonte: O autor (2019).

É simples observar que para N=100 o algoritmo MP possui um custo 10000, já o MPM apenas metade disso e o ME irrisórios 100. Para valores maiores essa discrepância se torna ainda mais significativa, como pode ser observado na curva de crescimento do gráfico a seguir:

Gráfico 1 – Gráfico demonstrativo da curva de crescimento do custo dos algoritmos.



Fonte: O autor (2019).

5.2 PROJETOS PARA O FUTURO

Analisando os gráficos da seção anterior, podemos assumir que o objetivo deste trabalho foi alcançado, uma vez que foi possível não somente incrementar o estado da arte para o problema das n-rainhas, mas como também desenvolver um algoritmo eficiente.

Produzir este trabalho foi uma aprendizagem, como também um desafio pessoal e observar este novo modelo de algoritmo para o problema das n-rainhas, faz crer que há novas descobertas a serem feitas.

Uma das metas para o futuro é identificar sistemas reais onde ocorram conjuntos de dados com características das n-rainhas, para que o algoritmo aqui desenvolvido possa então ser aplicado e demonstrar sua verdadeira eficiência.

REFERÊNCIAS

1. ABRAMSON, B.; YUNG, M. Divide and conquer under global constraints: a solution to the n-queens problem**. Journal of Parallel and Distributed Computing**, v. 6, p. 649-662, 1989.

2. AHRENS, W. **Mathematische unterhaltungen und spiele**, Leipzig: B.G. Teubner, 1901.

3. AMBRUS, G.; BARÁT, J. A contribution to queens graphs: a substitution method, **Discrete Mathematics**, v. 306, p. 1105-1114, 2006.

4. BELL, J.; STEVENS, B. A survey of known results and research areas for *n*-queens. **Discrete Mathematics**, v. 309, p. 1-31, 2009.

5. BEZZEL, F. Proposal of eight queens problem. **Berliner Schachzeitung**, v. 3, p. 363, 1848.

6. BURGER, A.; COCKAYNE, E.; MYNHARDT, C. Domination and irredundance in the queens' graph. **Discrete Mathematics**, v. 163, p. 47-66, 1997.

7. BURGER, A.; MYNHARDT, C. An upper bound for the minimum number of queens covering the $n \times n$ chessboard. **Discrete Applied Mathematics**, v. 121, p. 51-60, 2002.

8. CAMPBELl, P. Gauss and the eight queens problem, a study in miniature of the propagation of historical error. **Historia Mathematica**, v. 4, p. 397-404, 1977.

9. CHATHAM, R.; DOYLE, M.; MILLER, J.; ROGERS, A.; SKAGGS, R.; WARD, J. Algorithm performance for chessboard separation problems. **Journal of Combinatorial Mathematics and Combinatorial Computing**, v. 70, 2009.

10. CHATHAM, R.; DOYLE, M.; JEFFERS, R.; KOSTERS, W.; SKAGGS, R.; WARD, J. Centrosymmetric solutions to chessboard separation problems. **Bulletin of the Institute of Combinatorics and its Applications**, v. 65, 2012.

11. CLAPP, R.; MUDGE, T.; VOLZ, R. Solutions to the n-queens problem using tasking in Ada. **ACM SIGPLAN Notices**, v. 21, p. 99-110, 1986.

12. COCKAYNE, E.; HEDETNIEMi, S. On the diagonal queens domination problem. **Journal of Combinatorial Theory**: series A, v. 42, p. 137-139, 1986.

13. COCKAYNE, E.; SPENCER, P. On the independent queens covering problem. **Graphs and Combinatorics**, v. 4, p. 101-110, 1988.

14. COCKAYNE, E. Chessboard domination problems. **Discrete Mathematics**, v. 86, p. 13-20, 1990.

15. COURNIA, N. Chessboard domination on programmable graphics hardware. *In*: ANNUAL SOUTHEAST REGIONAL CONFERENCE, 44. **Proceedings** […] p. 62-67, 2006.

16. CRAWFORD, K. Solving the n-Queens problem using genetic algorithms. *In*: ACM/SIGAPP SYMPOSIUM ON APPLIED COMPUTING: TECHNOLOGICAL CHALLENGES OF THE 1990'S. **Proceedings** […] p. 1039-1047, 1992.

17. CULL, P.; PANDEY, R. Isomorphism and the n-Queens problem, **ACM SIGCSE Bulletin**, v. 26, p. 29-36, 1994

18. DRAA, A.; MESHOUl, S.; TALBi, H.; BATOUCHE, M. A quantum-inspired differential evolution algorithm for solving the n-queens problem. **The International Arab Journal of Information Technology**, v. 7, p. 21-27, 2010.

19. ERBAS, C.; SARKESHIK, S.; TANIK, M. Different perspectives of the n-queens problem. *In*: CSC '92: ACM ANNUAL CONFERENCE ON COMMUNICATIONS. **Proceedings** […] p. 99-108, 1992.

20. ERBAS, C.; TANIK, M.; ALIYAZICIOGLU, Z. Linear congruence equations for the solutions of the *n*-queens problem. **Information Processing Letters**, v. 41, p. 301-306, 1992.

21. ERBAS, C.; TANIK, M. Generating solutions to the *n*-queens problem using 2-circulants. **Mathematics Magazine**, v. 68, p. 343-356, 1995.

22. FOULDS, L.; JOHNSTON, D. An application of graph theory and integer programming: chessboard nonattacking puzzles. **Mathematics Magazine**, v. 57, n.3, p. 95-104, 1984.

23. GOLOMB, S.; BAUMERT, L. Backtrack Programming. **Journal of the ACM**, v. 12, p. 516-524, 1965.

24. GRAY, J. Is eight enough? the eight queens problem re-examined. **ACM SIGCSE Bulletin**, v. 25, p. 39-44,51, 1993.

25. GRINSTEAD, C.; HAHNE, B.; VAN STONE, D. On the queen domination problem. **Discrete Mathematics**, v. 86, p. 21-26, 1990.

26. HOFFMAN, E.; LOESSI, J.; MOORE, R. Constructions for the solution of the m-queens problem. **Mathematics Magazine**, v. 42, p. 66-72, 1969.

27. HOMAIFAR, A.; TURNER, J.; ALI, S. The n-queens problem and genetic algorithms. *In*: IEEE SOUTHEAST CONFERENCE, 1. **Proceedings** […] p. 262-267, 1992.

28. JING, H.; LIU, J.; CAI, Q. From *alife* agents to a kingdom of *n* queens. **Intelligent Agent Technology:** systems, methodologies, and tools, p. 110-120, 1999.

29. KALÉ, L. An almost perfect heuristic for the *n* nonattacking queens problem. **Information Processing Letters**, v. 34, p. 173-178, 1990.

30. KNUTH, Donald E. **The art of computer programming**, **volume 1**: fundamental algorithms. Redwood City, CA: Addison Wesley Longman Publishing Co., 1997.

31. MAAZALLAHI, R.; NIKNAFS, A.; ARABKHEDRI, P. A Polynomial-Time DNA computing solution for the n-queens problem. **Procedia:** social and behavioral sciences, v. 83, p. 622-628, 2013.

32. MANDZIUK, J.; MACUKOW, B. A neural network designed to solve the n-queens problem. **Biological Cybernetics**, v. 66, p. 375-379, 1992.

33. MANDZIUK, J. Solving the n-queens problem with a binary hopfield-type network. Synchronous and Asynchronous Model. **Biological Cybernetics**, v. 72, p. 439-446, 1995.

34. MONSKY, P. Problem E3162: superqueens, **The American Mathematical Monthly**, v. 93, n. 7, p. 566, 1986.

35. MONSKY, P. Problem E3162: superqueens. **The American Mathematical Monthly**, v. 96, n. 3, p. 258-259, 1989.

36. N-QUEENS bibliography: 336 references. Disponível em: http://liacs.leidenuniv.nl/~kosterswa/nqueens/. Acesso em: 26 fev. 2017.

37. NADEL, B. Representation selection for constraint satisfaction: a case study using *n*-queens. **IEEE Expert**, v. 5, p. 16-23, 1990.

38. NAUCK, F. Briefwechsel mit allen fr alle, **Leipziger Illustrierte Zeitung**, v. 377, p. 182, 1850.

39. NOGUCHI, W.; PHAM, C.-K. A proposal to solve n-queens problems using maximum neuron model with a modified hill-climbing term. *In*: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 6. **Proceedings** […] p. 2679-2682, 2006.

40. GENT, I. P.; JEFFERSON, C.; NIGHTINGALE, P., Complexity of n-queens completion. **Journal of Artificial Intelligence Research**, v. 59, p. 815-848, 2017.

41. PREUSBER, T.B.; ENGELHARDT, M.R. Putting queens in carry chains. **Journal of Signal Processing Systems**, n. 27, 2016.

42. REICHLING, M. A simplified solution of the n queens' problem. **Information Processing Letters**, v. 25, p. 253-255, 1987.

43. RIVIN, I.; ZABIH, R. A dynamic programming solution to the *n*-queens problem. **Information Processing Letters**, v. 41, p. 253-256, 1992.

44. ROLFE, T., Queens on a Chessboard: Making the Best of a Bad Situation. *In*: ANNUAL SMALL COLLEGE COMPUTING SYMPOSIUM, 28. **Proceedings** […] v. 28, p. 201-210, 1995.

45. SAN SEGUNDO, P. New decision rules for exact search in n-queens, **Journal of Global Optimization**: TBA, p. 1-18, 2011.

46. SHAGRIR, O. A neural net with self-inhibiting units for the *n*-queens problem. **International Journal of Neural Systems**, v. 3, p. 249-252, 1992.

47. SILVA, I.; SOUZA, A.; BORDON, M. A modified hopfield model for solving the n-queens problem, neural networks. *In:* IEEE-INNS-ENNS INTERNATIONAL JOINT CONFERENCE ON. **Proceedings** […] p. 509-514, 2000

48. SOSIC, R.; GU, J. A polynomial time algorithm for the n-queens problem. **ACM SIGART Bulletin**, v. 1, p. 7-11, 1990.

49. SOSIC, R.; GU, J. 3,000,000 queens in less than one minute. **ACM SIGART Bulletin**, v. 2, p. 22-24, 1991.

50. SPRAGUE, T., On the Different Non-Linear Arrangements of Eight Men on a Chess-board. *In:* EDINBURGH MATHEMATICAL SOCIETY. **Proceedings** […] v. 8, p. 30-43, 1889.

51. STONE, H.; STONE, J. Efficient Search Techniques --- An empirical Study of the N-Queens Problem, **IBM Journal of Research and Development**, v. 31, p. 464-474, 1987.

52. TOPOR, R., Fundamental Solutions of the Eight Queens Problem. **BIT Numerical Mathematics**, v. 22, p. 42-52, 1982.

53. VASQUEZ, M.; HABET, D. Complete and incomplete algorithms for the queen graph coloring problem. *In*: EUROPEAN CONFERENCE ON ARTFICIAL INTELLIGENCE, 16. **Proceedings** […] p. 226-230, 2004.

54. VASQUEZ, M. New results on the queens_n^2 graph coloring problem. **Journal of Heuristics**, v. 10, p. 407-413, 2004.

55. ZONGYAN, Q. Bit-vector encoding of n-queen problem, **ACM SIGPLAN Notices**, v. 37, p. 68-70, 2002.