Universidade Federal de Uberlândia - UFU
Bacharelado em Sistemas de Informação - Campus Monte Carmelo
João Vitor Pereira - 31711BSI024
Cesar Henrique - 31711BSI006

1 Introdução

Este trabalho aborda o problema das "n-rainhas", que está situado em um ambiente na qual temos que procurar uma solução que permita que uma quantidade N de rainhas seja disposta em um tabuleiro de tamanho N por N, na qual nenhuma peça ataque outra. Levando isto em consideração, é necessário entendermos a sua movimentação e que, além disso, seja definido em quais situações, dentro de um jogo de xadrez, é determinado que uma rainha está atacando a outra.

Em um jogo de xadrez, existem 6 peças diferentes e com comportamentos característicos, na qual a rainha é considerada a peça mais valiosa por conseguir ter uma possibilidade maior de movimentação no tabuleiro, conseguindo se mover livremente (quantas casas desejar) por toda a sua coluna, linha e até mesmo nas diagonais, como é mostrado na Figura 1. Afirmar que uma rainha está atacando a outra, significa que existe duas ou mais peças em uma mesma linha, coluna ou diagonal. Desta forma, posicionar 8 rainhas em um tabuleiro de xadrez de tamanho 8 por 8, não se torna uma tarefa trivial, pois a escolha da casa em que uma dessas peças irá ocupar, afeta diretamente no posicionamento da próxima rainha no tabuleiro e, além disso, pensar em uma solução para este problema se torma algo bastante complexo quando consideramos valores grandes para N.

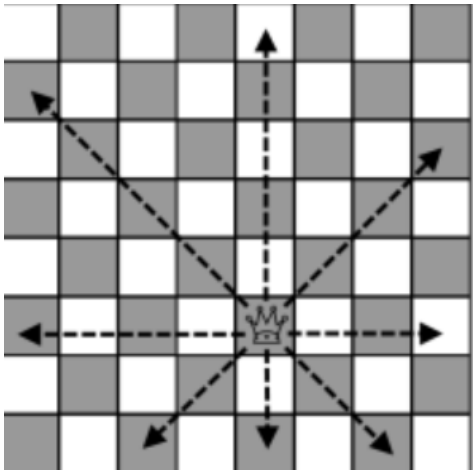

Figura 1: Movimentação da rainha em um tabuleiro de xadrez

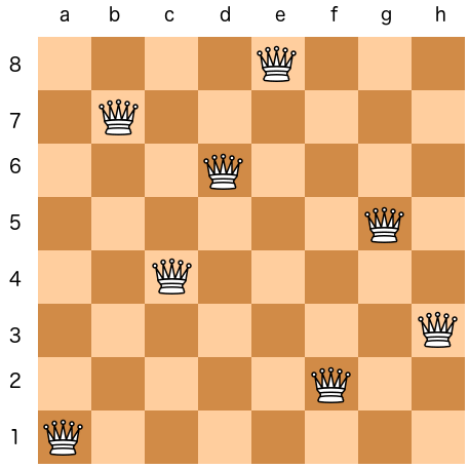

Figura 2: Exemplo de solução para um tabuleiro de tamanho N = 8

Partindo para a abordagem do problema, iremos utilizar o algoritmo Simulated Annealing, que representa uma abstração do processo de aquecimento e resfriamento de metais, denominado em inglês Annealing. Basicamente o processo aquece o metal até uma temperatura bem alta para depois resfriá-lo lentamente, conseguindo um material mais uniforme e com melhores propriedades. Fazendo uma analogia para o problema das n-rainhas, o algoritmo em questão será executado iterativamente buscando por soluções cada vez melhores ao passo que a temperatura será gradativamente diminuída a cada iteração, e a medida que esta temperatura se aproxima de zero, o comportamento do algoritmo tende demonstrar resistência ao selecionar estados com pior quualidade que o estado atual. Além disso, por se tratar de um problema de minimização, definimos que um estado é melhor que o outro pela quantidade de conflitos que ele possui, ao efetuar o cálculo da função objetivo, daremos prioridade para aquela configuração de tabuleiro que retornar um número mais próximo de zero, indicando uma baixa quantidade de rainhas se atacando.

O fluxo do programa inincia com a geração de um tabuleiro de tamanho N, representando a solução inical do problema, com valores randômicos e, após exibir a quantidade de conflitos existentes na mesma, o algoritmo irá iniciar o processo do Simulated Annealing com os parâmetros definidos no código. O padrão é número de rainhas igual a 8, temperatura inicial de 100 e uma taxa de resfriamento de 0.9999, podendo ser livremente alterados.

O processo será executado em uma estrutura de repetição, sendo que a cada iteração o algoritmo multiplicará a temperatura pela taxa de resfriamento até que seja obtido o máximo global (tabuleiro sem conflitos) ou que a temperatura seja inferior a 0.0000000001. Após um leve resfriamento, será gerado um array contendo dois valores aleatórios distintos, que representam índices de duas colunas do tabuleiro que serão trocadas. A troca é realizada atribuíndo o valor de uma coluna para outra, ou seja, trocando duas rainhas de linha, conforme mostra a Figura 3, neste exemplo foram escolhidos os índices 1 e 2, cujos valores das linhas são 6 e 2 respectivamente.

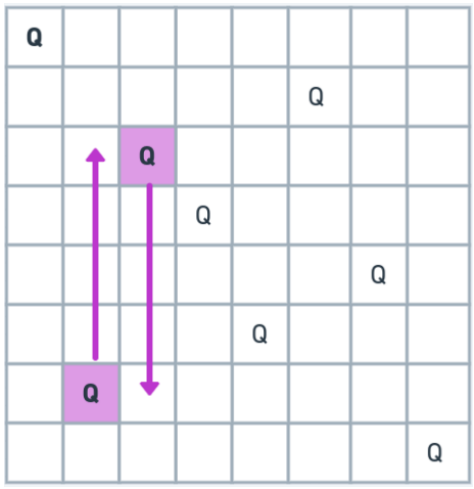

Figura 3: Escolhendo dois indices aleatórios de colunas

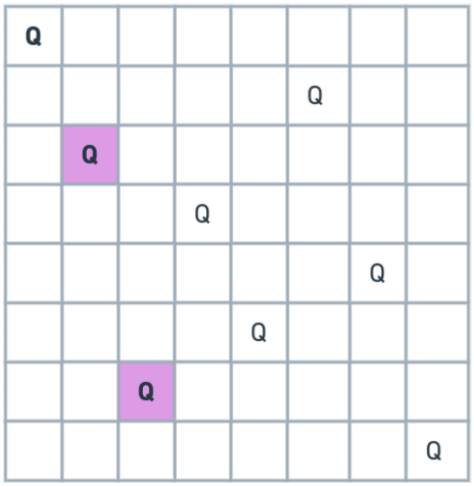

Figura 4: Trocando a linha dos índices selecionados

Após gerar o sucessor da solução inicial, efetuando a troca de colunas explicada anteriormente, é realizado o cálculo de variação entre a qualidade da solução inicial e a do sucessor. Se a função objetivo retornar um valor melhor para o sucessor, ele é escolhido para ser a solução desta iteração, caso o contrário este sucessor ainda possui uma chance de ser escolhido baseado na temperatura atual do sistema. Quando maior a temperatra se encontra, mais chance o sucessor possui para ser escolhido, e a probabilidade é dada pela seguinte fórmula: **exp(-variação/temperatura)**. É gerado um valor aleatório entre zero e um, e comparado se este valor gerado é menor que o resultado da fórmula anterior, de forma que quando a variação é muito grande ou o valor da temperatura é muito baixo, o valor retornado tende a ser mais próximo de zero, sendo muito difícil escolher o sucessor. Logo, a chance de se escolher o sucessor quando seu número de conflitos é maior que a solução atual, é baseada na temperatura do sistema e na variação entre a avaliação desses estados.

Uma vez escolhido o estado da iteração baseado nos critérios apresentados anteriormente, o algoritmo será executado até que a temperatura seja bem próxima a zero, ou que o número de conflitos do estado atual seja igual a zero.

2 Problema

2.1 Abordagem do problema

Quando pensamos no problema das n-rainhas é preciso abstrair um tabuleiro com várias destas peças posicionadas em linhas e colunas, tendo como objetivo, encontrar uma configuração de forma que elas não se ataquem. Quais são as condições possíveis que temos que nos preocupar para conseguir prevenir que uma rainha ofereça perigo a outra? A resposta é bem simples, temos que olhar as direções que ela pode se movimentar e verificar se nestas direções existe outra peça, com isso chegamos nas seguintes situações que apresentam uma ameaça:

- **Duas ou mais rainhas alocadas na mesma linha.**
- **Duas ou mais rainhas alocadas na mesma coluna.**
- **Duas ou mais rainhas alocadas na mesma diagonal principal.**
- **Duas ou mais rainhas alocadas na mesma diagonal secundária.**

Além disso, nas seções seguintes os termos "diagonal do tabuleiro"e "diagonal da rainha"serão bastante utilizados. Esses dois termos vão ser usados para referenciar posições no tabuleiro, assim como é mostrado na Figura 5 e seu entendimento é de fundamental importância.

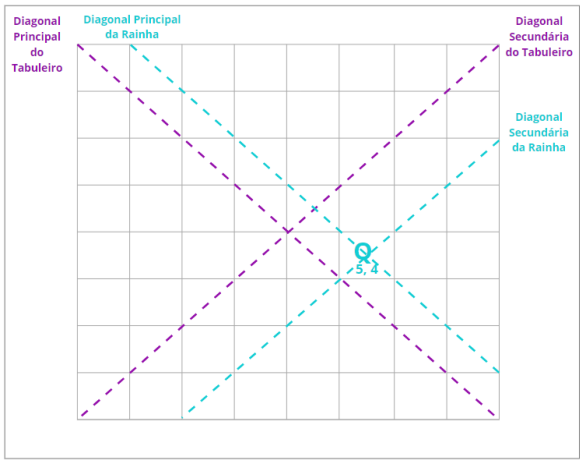

Figura 5: Diferenciação das diagonais

A partir do conhecimento das situações que apresentam conflitos entre as rainhas e referênciais de posicionamento, precisamos definir uma boa heurística para este problema juntamente com um teste de meta que nos diga se chegamos no ótimo global ou não, na qual as próximas seções abordarão esses tópicos e outros.

2.2 Estado inicial

O estado inicial deste algoritmo consiste em basicamente gerar uma configuração aleatória de rainhas no tabuleiro, na qual pode ser realizada de diversas maneiras, neste trabalho, esse arranjo inicial das peças foi gerado através de dois passos. o primeiro foi gerar uma lista de N posições contendo valores de 0 até N - 1 em ordem crescente. Se este array, gerado fosse usado para ser a configuração inicial do problema, as rainhas estariam dispostas em todas as posições da diagonal principal do tabuleiro, de forma que todas estariam em conflito, conforme mostra a Figura 6. O segundo passo foi embaralhar essa lista, trocando os valores dos índices de lugar, sem modificar a sua quantidade de elementos ou algo do tipo, apenas encontrando uma ordem aleatória para os elementos da lista, que vão de 0 até N -1, mas agora, não necessáriamente, os elementos desse vetor estão em ordem crescente, como é exemplificado pela Figura 7.

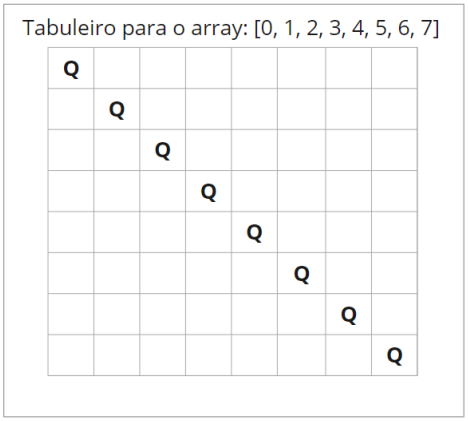

Figura 6: Tabuleiro após primeiro passo de geração de estado inicial

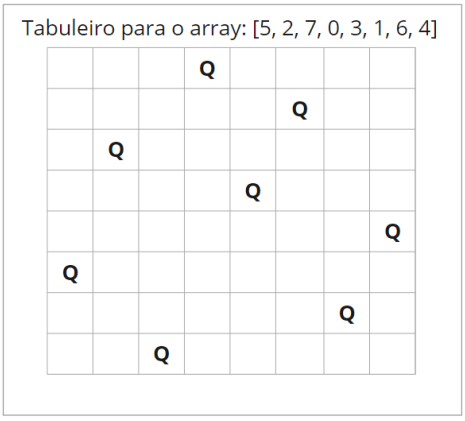

Figura 7: Tabuleiro após segundo passo de geração de estado inicial

A lista criada é interpretada pelo algoritmo de uma forma bastante específica, na qual o índice da posição representa a coluna e, o valor desta posição, representa a linha. A principal vantagem de gerar o tabuleiro usando

esta metodologia está no fato de que não é possível gerar combinações na qual exista uma ou mais rainhas na mesma coluna ou na mesma linha, ou seja, os únicos conflitos podem ser ocasionados apenas por rainhas que se atacam pela diagonal e, tal estratégia, otimiza bastante a execução do algoritmo.

2.3 Função Objetivo e Teste de Meta

Uma vez que o tabuleiro é gerado, é necessário calcular o custo desta solução inicial através de uma função objetivo, que é uma forma númerica de representar a qualidade do estado atual. Uma vez que não é possível posicionar duas ou mais rainhas na mesma linha ou coluna, a função objetivo precisa determinar apenas a quantidade de rainhas que se atacam pela diagonal. Dessa forma foi pensado em uma maneira de validar a quantidade de rainhas que existem na mesma diagonal principal e secundária da peça que estamos analisando no momento. Através de uma iteração para cada rainha do tabuleiro, convergimos cada uma das suas duas diagonais referente a posição em que a mesma se encontra para um índice em comum que é utilizado para validar as próximas peças de forma incremental.

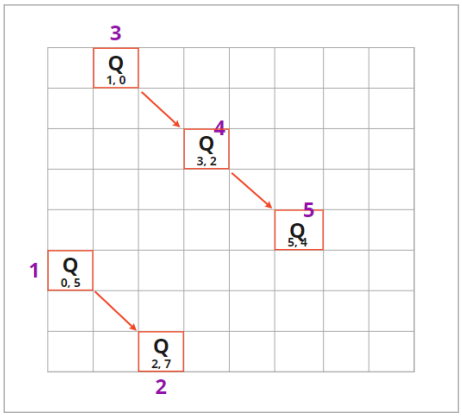

Figura 8: Conflitos em duas diagonais principais

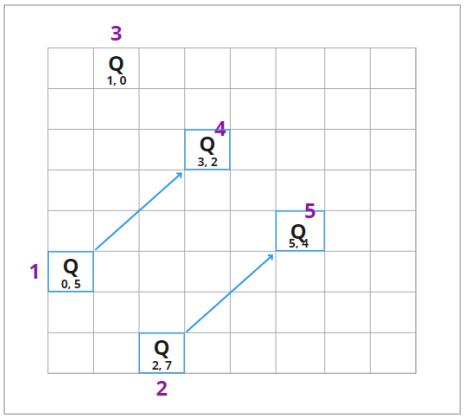

Figura 9: Conflitos em duas diagonais secundárias

Na Figura 8, duas diagonais principais estão sendo utilizadas, sendo que na localizada mais próxima à diagonal principal do tabuleiro, com deslocamento de uma casa positiva de distância, as rainhas 3, 4 e 5 estão se atacando, e as rainhas 1 e 2 também estão se atacando em outra diagonal, com 5 casas de distância negativa em relação a diagonal principal do tabuleiro. Na Figura 9, podemos perceber que duas diagonais secundárias estão em uso, na qual as rainhas 1 e 4 estão se atacando em uma diagonal com duas casas de deslocamento negativo em relação à diagonal secundária do tabuleiro, e, as rainhas 2 e 5 estão se atacando em uma diagonal com um deslocamento positivo de duas casas.

Sempre que for subtraído o índice da coluna da rainha menos o valor de sua linha, encontraremos um valor x tal que $\{-N+1 \leq x \leq N-1\}$, que representa o deslocamento entre a diagonal principal do tabuleiro e a diagonal principal que a rainha está ocupando. Ademais, ao somar esses dois valores e subtrair do resultado por (N - 1), encontraremos um valor y com a mesma representação, mas desta vez para a diagonal secundária do tabuleiro em relação à peça que esta sendo analisada no momento, tal que $\{-N+1 \leq y \leq N-1\}$.

Após encontrar esse índice da diagonal principal e secundária da rainha em relação ao tabuleiro, devemos mapear e incrementar o valor desse índice em um dicionário para diagonais principais e outro para secundárias, existindo duas situações possíveis: o índice já existia no dicionário ou não. Na primeira situação essa chave recerá o valor 1, na segunda situação a chave receberá o valor atual somado de 1. Dessa forma, ao finalizar a validação de todas as peças do tabuleiro, existirão índices que possuem apenas o valor 1, indicando que naquela diagonal está presente uma única rainha, e existirão índices com valores maiores quue 1, indicando que naquela diagonal mapeada existe mais de uma rainha e, portando, o tabuleiro apresenta conflitos. Observe nas Figuras 10 e 11 a seguir um exemplo para o tabuleiro criado anteriormente com 5 rainhas.

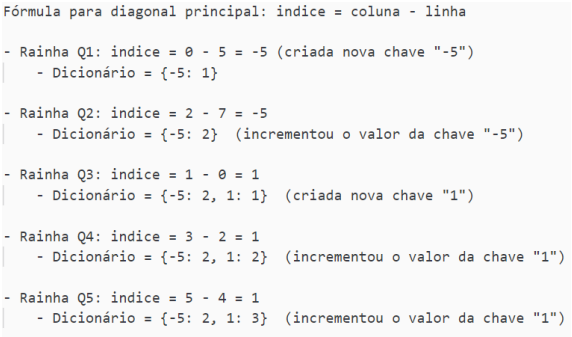

Figura 10: Geração das chaves do dicionário que mapeia conflitos na diagonal principal

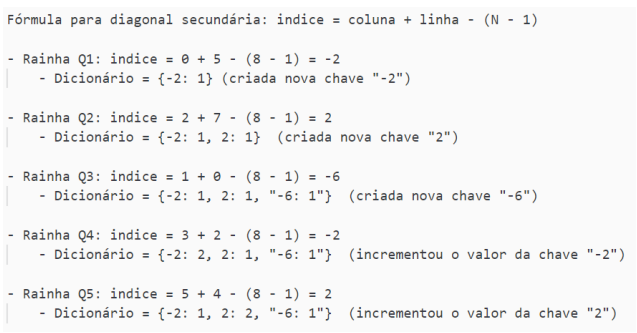

Figura 11: Geração das chaves do dicionário que mapeia conflitos na diagonal secundária

A quantidade total de conflitos no tabuleiro é obtida basicamente somando os valores de todas as chaves dos dois dicionários criados, um que mapeia os conflitos nas diagonais principais e outro nas diagonais secundárias. Porém uma soma simples geraria um valor de conflitos incorreto, pois o valor contido em cada chave desse dicionário é a quantidade de rainhas presente em uma determinada diagonal e não a quantidade de ataques causados entre as mesmas. Essa é uma diferença de fundamental importância, pois, por exemplo, caso exista 4 rainhas, R1, R2, R3 e R4, em uma mesma diagonal, a quantidade de conflitos entre essas peças é um total de 6, que é calculado da seguinte forma:

- R1 ataca R2, R3 e R4
- R2 ataca R3 e R4
- R3 ataca R4

Portanto, a fórmula que calcula a quantidade de conflitos em uma diagonal, dada uma quantidade Q de rainhas presentes na mesma é obtida pela seguinte fórmula: $(Q-1) \cdot Q/2$. A partir disso, para obtermos a quantidade de conflitos no tabuleiro, e consequentemente a função objetivo deste problema, precisamos aplicar a fórmula descrita anteriormente em cada chave dos dicionários e somar esses resultados.

Por se tratar de um problema de minimização, cujo objetivo é encontrar um arranjo de peças no tabuleiro que não se conflitem em nenhuma direção, podemos afirmar que o teste meta deste problema é verificar se a função objetivo retornou o valor zero. Caso essa situação ocorrer, dizemos que o algoritmo encontrou o máximo global, ou seja, encontrou a melhor solução em todo o espaço de soluções possíveis, porém esse estado nem sempre é tão simples de ser obtido, necessitando de uma série de melhorias na representação do problema ou na escolha dos parâmetros do mesmo.

2.4 Ações

As possíveis ações compreendidas pelo algoritmo se baseiam principalmente na troca de rainhas do tabuleiro, na qual pode ser realizada de diversas maneiras. Neste trabalho optamos por escolher apenas duas rainhas e trocar as linhas que elas ocupam, de forma a gerar um novo arranjo e, consequentemente, um novo estado para o algoritmo. A metodologia para a escolha de duas rainhas é realizada gerando dois números aleatórios entre zero e N-1, que serão usados para informar quais colunas terão suas linhas trocadas.

Alisando melhor as Figuras 3 e 4 podemos perceber que no estado inicial foram escolhidas a segunda e terceira coluna do tabuleiro, e em seguida as linhas nas quais as rainhas estavam foram trocadas. Esta ação impactou diretamente na quantidade de conflitos, pois antes da troca a rainha alocada na terceira coluna estava conflitando com a rainha presente na 1ª, 4ª e 8ª coluna. Após a troca de linhas, a rainha que agora está na segunda coluna está conflitando somente com a 5º coluna, enquanto que a terceira coluna não está mais apresentando conflitos.

A partir disso, podemos perceber que a nova solução (Figura 4) apresenta uma configuração melhor que a anterior(Figura 3), passando para um estado com menos conflitos entre as rainhas do tabuleiro. Porém é importante ressaltar que nem sempre isso acontece, ou seja, da mesma forma que a nova solução pode apresentar resultados melhores, ela também pode apresentar um aumento de conflitos no momento em que faz a troca, pois os dois índices que são usados nessa operação são gerados de forma aleatória, cujo o único critério é que sejam números diferentes.

As Figuras 12 e 13 exemplificam outra situação de geração de um sucessor, realizando uma troca partindo da mesma configuração da Figura 3, porém foi escolhida as colunas 1 e 5, ou seja, a 2ª e a 6ª. Após a troca, temos a rainha da 2ª coluna conflitando com as rainhas da 1ª, 3ª,4ª e 8ª colunas respectivamente, e a rainha da 6ª coluna está conflitando com a rainha da 5ª coluna, exemplificando que nem sempre a solução sucessora possui um desempenho melhor que a anterior.

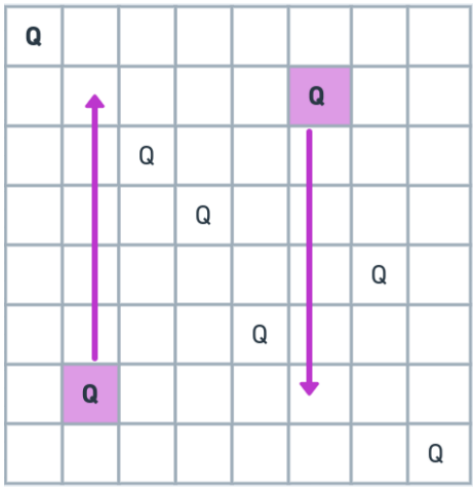

Figura 12: Escolhendo dois indices aleatórios de colunas

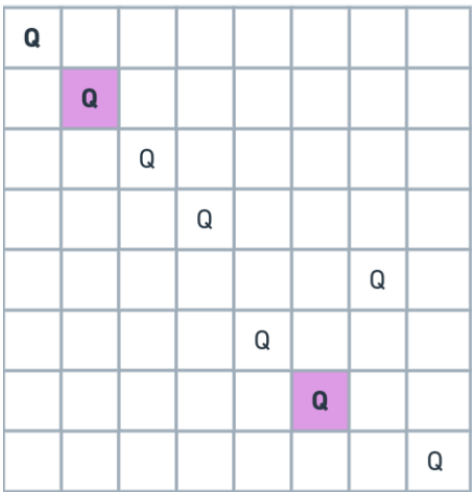

Figura 13: Trocando a linha dos índices selecionados

O motivo para a adoção desta metodologia se baseia principalmente na escolha de uma opção com baixo custo de processamento, visto que o algoritmo realizará diversas iterações, pensar em uma alternativa mais minimalista parecia fazer mais sentido. Porém, é importante ressaltar que é possível utilizar outras abordagens que se baseiam em uma escolha mais bem trabalhada, com o objetivo de diminuir a quantidade de iterações do algoritmo, ao custo de realizar mais operações computacionais.

2.5 Modelo de transição

A transição de um estado para outro no algoritmo Simulated Annealing é algo bastante característico, e inclusive é o seu principal diferencial para outros algoritmos similares, como o Hill Climbing. Através de uma possibilidade de escolher um estado pior do que o atual através do uso da temperatura e taxa de resfriamento, o Simulated Annealing consegue escapar de ótimos locais, que é uma situação em que o Hill Climbing possui uma certa limitação.

Após gerado dois números aleatórios para serem os índices das colunas nas quais suas rainhas trocarão de linhas, e a troca é realizada, conseguimos um novo estado, que será chamado de sucessor. O próximo passo do algoritmo, consiste em verificar qual estado possui uma melhor qualidade, e para isso é realizado uma subtração do valor da função objetivo do sucessor menos o valor da funçao objetivo para a solução atual. Caso essa variação seja negativa, quer dizer que o sucessor possui uma melhor configuração em seu tabuleiro, e será escolhido para ser o novo estado atual do sistema. Caso contrário, ou seja, a variação é positiva ou igual a zero, significa que o estado atual é melhor ou igual ao sucessor em termos de heurística, mas ainda assim existe uma chance do algoritmo transicionar o estado para o sucessor.

Essa chance é dada em função da variação e da temperatura do sistema, visto que a cada iteração essa temperatura sofre uma diminuição, o comportamento do programa se baseia em escolher com bastante recorrência o sucessor quando a temperatura está alta e ter um comportamento mais seletivo a medida que a mesma se torna cada vez mais próxima a zero. A fórmula que é utilizada para obter essa probabilidade é dada por: **exp(-variação/temperatura)**, na qual sempre que ela for utilizada, a variável "variação"terá um valor positivo, o que ocasionará em uma exponenciação com um expoente negativo, e quanto maior for o valor deste expoente menor será o número gerado pela fórmula. Sabendo disso, podemos afirmar que quanto menor for a temperatura, maior será o expoente, e consequentemente o resultado será mais próximo de zero, e o mesmo se aplica para quando o valor da variação é muito alto.

Após obter o resultado da fórmula anterior, é comparado se esse resultado é menor que um número gerado aleatóriamente entre zero e um, conseguindo assim uma probabilidade de se escolher o sucessor baseado no valor da variação e da energia atual do sistema. Uma vez que o valor randômico gerado é menor que o resultado da fórmula, o algoritmo atribui o estado do sucessor como o estado atual, e o programa computa essa sequência de comandos iterativamente até que um dos dois critérios de parada mencionados anteriormente sejam satisfeitos.

3 Algoritmo

3.1 Desenvolvimento do algoritmo e Bibliotecas Utilizadas

O algoritmo utilizado neste trabalho foi desenvolvido em Python 3, devido a popularidade que a própria linguagem possui para resolver problemas relacionados ao deste trabalho, além de possuir varias funções que facilita o desenvolvimento do mesmo, sendo bem mais fácil realizar tarefas como a criação do tabuleiro, cálculo de ameaças, exibição do tabuleiro além da própria função que executa o Simulated Annealing. Além disso foram utilizadas as seguintes bibliotecas:

- Random: Foi utilizada com o objetivo de gerar números aleatórios e embaralhar arrays.
- Math: A biblioteca math nos permite utilizar a função exp (exponencial), que é bastante importante para a execução do Simulated Annealing sendo responsável por determinar a probabilidade de escolher uma solução pior que a melhor encontrada até então.
- Time: Utilizada para exibir o tempo gasto durante a execução do algoritmo.
- Copy: Utilizada para copiar um array para outra variável sem se preocupar com referências de memória.

Todo o código é composto por 6 funções principais que tem a responsabilidade de dividir as responsabilidades e possíbilitar um código mais limpo e de melhor manutenção, sendo essas:

- main: Função principal a ser executada, sendo esta que controla o início e fim do fluxo de execução.
- create_board(n): Função responsável por criar um tabuleiro de tamanho de tamanho n x n com n rainhas posicionadas de forma aleatória, sendo que ele já possibilita uma criação randômica sem que exista rainhas na mesma coluna e linha.
- evaluate_solution(board): Função objetivo do algoritmo, responsável por calcular a quantidade de conflitos no tabuleiro representado pelo array "board"passado como parâmetro.
- calc_number_of_conflics_in_diag(n): Função responsável por calcular o número de conflitos em uma diagonal com "n"rainhas.
- simulated_annealing(s, t, tax): Função responsável por executar o algoritmo do Simulated Annealing, cuujos parâmetros é solução inicial "s", temperatura "t"e taxa de resfriamento "tax".
- display(board): Função responsável por exibir no console o modelo visual da tabela, desenhando um tabuleiro de rainhas caso N seja menor que 40, caso o contrário o tabuleiro não é exibido por ser muito grande.

3.2 Análise de resultados

Neste módulo será efetuado alguns testes utilizando o algoritmo desenvolvido em Python com objetivo de explicitar a importância e o impacto da escolha dos parâmetros iniciais do problema. Foi realizado diversos testes alterando a combinação entre as temperaturas 1, 10 e 100 juntamente com as taxas de resfriamento 0.9, 0.99 e 0.999, para tabuleiros de tamanho 8, 30, 100 e 1000, na qual neste último foi realizado mais testes com outras parametrizações. Porém, antes de analisar os resultados providos pelas combinações desses valores, é importante ressaltar algumas características do algoritmo que serão abordadas a seguir.

Número de Rainhas	8	30	100	500	1000	10000
Número Inicial de Conflitos	5	25	71	322	657	6712
Taxa de resfriamento	0.99	0.99	0.99	0.99	0.99	0.99
Número Interações	381	759	2751	2751	2751	2751
Melhor solução	0	0	5	62	175	2751

Figura 14: Resultados obtidos testando uma taxa de resfriamento de 0.99

Na figura 14 podemos observar os resultados obtidos utilizando uma taxa de resfriamento de 0.99, e ressaltar alguns aspectos interessantes. O primeiro ponto a ser observado é que o algoritmo foi capaz de encontra a solução ótima somente para problemas com até 30 rainhas, isso devido ao fato de que com uma temperatura inicial de 100 e uma taxa de 0.99 o máximo de interações que o algoritmo pode realizar é de 2751, explicitando que é necessário mais iterações e um refinamento maior dos estados para conseguir atingir o máximo global para tabuleiros de tamanhos superiores a 30.

Número de Rainhas	8	30	100	500	1000	10000
Número Inicial de Conflitos	5	16	49	305	663	6724
Taxa de resfriamento	1 - (1/10**4)	1 - (1/10**4)	1 - (1/10**4)	1 - (1/10**4)	1 - (1/10**4)	1 - (1/10**4)
Número Interações	155	61593	63651	96098	161550	276298
Melhor solução	0	0	0	0	0	291

Figura 15: Resultados obtidos com taxa de 1 - (1/10**4)

Na figura 15 temos a mesma temperatura inicial de valor 100, porém desta vez será utilizada uma taxa de resfriamento de 1 - (1/10**4), ou seja, de 0.9999. Neste segundo teste foi obtido resultados bastantes positivos para os casos com rainhas acima de 30, pois com uma taxa de resfriamento menor foi possível a realização de mais iterações, obtendo melhores resultados para tabuleiros com mais rainhas. Outro ponto que podemos destacar, é que o algoritmo saiu de 2751 para 276298 iterações, e isso quase possibilitou encontrar a solução ótima para 10000 rainhas, logo podemos observar o quanto a taxa de resfriamento impacta na quantidade de iterações do algoritmo e, consequentemente, na exploração do espaço de estados do problema.

Outro ponto que podemos analisar é a relação entre a temperatura e a taxa de resfriamento, para isso agora será aborado os testes utilizando várias combinações de valores para os parâmetros do problema, sendo executadas sobre diferentes temperaturas e taxas de resfriamento, e com isso vamos poder ter uma base mais sólida do impacto de cada um desses valores no resultado final.

Temperatura	Taxa de Resfriamento	Qte. Interações	Tempo de Execução	Qtde Conflitos
1	0.9	65	0.005 s	0
1	0.99	296	0.006 s	0
1	0.999	111	0.004 s	0
10	0.9	171	0.004 s	0
10	0.99	56	0.008 s	0
10	0.999	94	0.006 s	0
100	0.9	34	0.003 s	0
100	0.99	27	0.006 s	0
100	0.999	19	0.009 s	0

Figura 16: Resultados obtidos com 8 rainhas

Podemos perceber na figura 16 que com um tamanho relativamente pequeno para o tabuleiro, como o caso de n=8, todas as combinações entre os valores conseguiram obter o máximo global, explicitando que a combinação

desses valores não precisa de um minunciosa análise para conseguir obter a solução ótima.

Temperatura	Taxa de Resfriamento	Qte. Interações	Tempo de Execução	Qtde Conflitos
1	0.9	220	0.089 s	3
1	0.99	562	0.102 s	0
1	0.999	2544	0.147 s	0
10	0.9	242	0.085 s	3
10	0.99	1048	0.094 s	0
10	0.999	5973	0.216 s	0
100	0.9	264	0.101 s	2
100	0.99	2751	0.146 s	1
100	0.999	5972	0.212 s	0

Figura 17: Resultados obtidos com 30 rainhas

No caso da figura 17 é possível perceber que um valor mais baixo para a taxa de resfriamento está impossibilitando que o algoritmo encontre uma solução ótima, mesmo a temperatura variando entre 1, 10 e 100. A partir disso, temos um primeiro indício que este parâmetro possui um maior impacto na performance do algoritmo em encontrar soluções ótimas, ao contrário do valor inicial da temperatura propriamente dita, mostrando a importância de utilizar uma taxa de resfriamento adequada para o ambiente. Quando falamos de taxa de resfriamento adequada, temos que pensar na taxa que mais se adequa com a magnitude do problema e com o ganho de perfomance do mesmo.

Temperatura	Taxa de Resfriamento	Qte. Interações	Tempo de Execução	Qtde Conflitos
1	0.9	220	0.014 s	24
1	0.99	2293	0.134 s	2
1	0.999	18067	1.136 s	0
10	0.9	242	0.026 s	21
10	0.99	2522	0.183 s	4
10	0.999	9019	0.554 s	0
100	0.9	264	0.015 s	15
100	0.99	2751	0.164 s	4
100	0.999	10233	0.610 s	0

Figura 18: Resultados obtidos com 100 rainhas

Podemos perceber na figura 18, que os únicos casos em que o algoritmo conseguiu achar uma solução ótima foi quando a taxa de resfriamento estava como 0.999, independentemente do valor inicial da temperatura, sendo assim possível afirmar e constatar novamente a importância de uma boa parametrização desta variãvel. Assim como no resfriamento de metais em um ciderúrgica, um resfriamento mais lento permite ao nosso algoritmo executar mais iterações e com isso possibilita uma maior exploração do espaço de estados do problema, tendendo a desenvolver uma solução com uma combinação de peças que ocasionam em menos conflitos.

Temperatura	Taxa de Resfriamento	Qte. Interações	Tempo de Execução	Qtde Conflitos
1	0.9	220	0.140 s	506
1	0.99	2293	1.588 s	207
1	0.999	23016	14.695 s	33
10	0.9	242	0.165 s	511
10	0.99	2522	1.797 s	190
10	0.999	25317	16.253 s	23
100	0.9	264	0.160 s	557
100	0.99	2751	1.684 s	187
100	0.999	27619	17.577 s	30

Figura 19: Resultados obtidos com 1000 rainhas

Durante esta análise, foi constatado a importância que a taxa de resfriamento possuí dentro do algoritmo, porém em nenhum ambiente de teste houve um caso que não houvesse tido uma solução com custo 0, porém ao efetuar testes com um ambiente de 1000 rainhas, os resultados obtidos não foram satisfatórios, como se pode observar na figura 19.

Ao analisar os ambientes de testes com 8, 30, 100 e 1000 rainhas podemos observar que houve uma gradativa diminuição de ocorrência de soluçÕes ótimas, na qual no primeiro tivemos todas os casos testes resultando em soluções de custo 0, enquanto que no último não foi encontrada. O algoritmo com uma taxa de resfriação x e uma temperatura inicial y possuí um limite de interações, e caso o problema seja grande demais, a parametrização inicial não possibilita o algoritmo a explorar todo o espaço de estados até encontrar um estado de custo zero, sendo necessário ajustar os parâmetros iniciais para se obter uma solução ótima.

Tendo em vista que é necessário aumentar o número de interações, qual seria o meio mais vantajoso de se fazer isso? aumentando a temperatura inicial, ou diminuíndo a taxa de resfriamento? Essa resposta será obtida através de um novo ambiente de testes com 1000 rainhas, mas desta vez será adicionado um novo valor para a temperatura e para a taxa de resfriamento, de 1000 e 0.9999, respectivamente.

Temperatura	Taxa de Resfriamento	Qte. Interações	Tempo de Execução	Qtde Conflitos
1	0.9	220	0.190 s	508
1	0.99	2293	1.821 s	191
1	0.999	23016	20.188 s	26
1	0.9999	127986	114.9279	0
10	0.9	242	0.195 s	497
10	0.99	2522	2.001 s	167
10	0.999	25317	21.489	30
10	0.9999	120693	109.023 s	0
100	0.9	264	0.228 s	550
100	0.99	2571	2.366 s	195
100	0.999	27619	24.092 s	21
100	0.9999	234771	204.893 s	0
1000	0.9	286	0.217 s	519
1000	0.99	2980	2.639 s	189
1000	0.999	29920	25.659 s	30

Figura 20: Resultados obtidos com 1000 rainhas em ambiente modificado

Novamente é notório que a taxa de resfriamento possuí uma maior importância no resultado final, como é mostrado na figura 20 o algoritmo só obteve resultados com custos 0 em casos de testes com a taxa de resfriamento mais baixa de 0.9999, sendo que o aumento da temperatura não teve uma influência direta nesse quesito, como podemos observar na última linha desta tabela, na qual com um valor mais alto de temperatura e mais alto de taxa de resfriamento, não foi possível obter o máximo global do problema. Ao contrário disso, é muito provável que

essa configuração de parâmetros ocasionou em um ótimo local quando a temperatura do sistema estava bem baixa, sendo quase impossível sair dessa situação.

4 Conclusão

Analisando o comportamento do algoritmo, é possível perceber que sua principal característica está na possibilidade de seleçao de um estado sucessor que possui uma qualidade pior que o estado atual. Tal capacidade é uma estratégia bastante pertinente para escapar de ótimos locais, problema que o algoritmo Hill Climbing possui certa dificuldade. Além disso, é importante ressaltar que esse comportamento se baseia na energia atual do sistema, na qual ao passo que a tempetura diminui, essa chance de selecionar um vizinho pior é reduzida considerávelmente, como é mostrado na Figura 21.

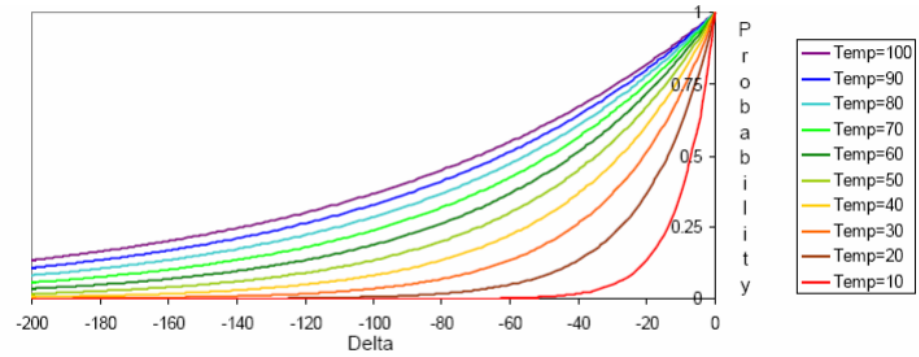

Figura 21: Probabilidade da solução ser aceita

A Figura 21 representa a probabilidade de escolha do novo estado ser aceito de acordo com a temperatura atual e sua variação em relação ao estado atual, logo se analisarmos o gráfico da direita para esquerda é possível perceber que quando a variação possui um valor alto, ou seja, mais próximo de zero, a probabilidade de uma solução ser aceita é extremamente alta, ao passo que a temperatura também se mantém em escalas elevadas quando a chance também é favorável à escolha do sucessor. Porém é possível notar que a medida que o valor de Delta se torna mais negativo e a medida que a energia do sistema é reduzida, a probabilidade se torna mais próxima de zero.

É importante ressaltar que a temperatura e a taxa de resfriamento tem um impacto muito grande na performance do algoritmo e que existem diversos estudos que abordam uma melhor escolha desses dois parâmetros. Se escolhermos, por exemplo, uma taxa de resfriamento mais próxima do valor 1 combinada, ou não, com uma temperatura maior, o programa tenderá a encontrar melhores resultados ao custo de demorar mais tempo para isso, pois será ocasionado uma maior resistência no quesito de transiçao de estados. Porém a escolha de um valor muito baixo ocasionará em uma execução mais rápida ao passo que a qualidade da solução poderá ser prejudicada negativamente. Sendo assim, é necessário encontrar um valor harmônico entre a taxa de resfriamento e a temperatura inicial levando em consideração as características do problema que estamos lidando.

Todo o código do trabalho pode ser encontrado no seguinte repositório do GitHub: `https://github.com/Cardoso-CHM/n-queens-simulated-annealing`