



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS SOBRAL
ENGENHARIA DA COMPUTAÇÃO
INTELIGÊNCIA COMPUTACIONAL

DENILSON GOMES VAZ DA SILVA

BUSCA EM FEIXE LOCAL PARA RESOLVER O PROBLEMA
DAS RAINHAS NAO ATACANTES

SOBRAL
2018

1. INTRODUÇÃO.....	2
2. REPRESENTAÇÃO.....	2
3. IMPLEMENTAÇÃO.....	2
4. RESULTADOS E COMPARAÇÃO COM BRUTE FORCE.....	7

1. INTRODUÇÃO

O problema das k rainhas não atacantes consiste em conseguir dispor k rainhas em um tabuleiro de k linhas por k colunas de forma que não haja nenhum ataque entre às mesmas. Em um tabuleiro convencional, de 8 linhas por 8 colunas, é possível colocar 8 rainhas de 92 maneiras diferentes.

Nesta solução, foi implementado o algoritmo de busca *Local Beam Search* (*Busca em Feixe Local*), e seu desempenho foi comparado com o desempenho do algoritmo *Brute Force* (*Força Bruta*).

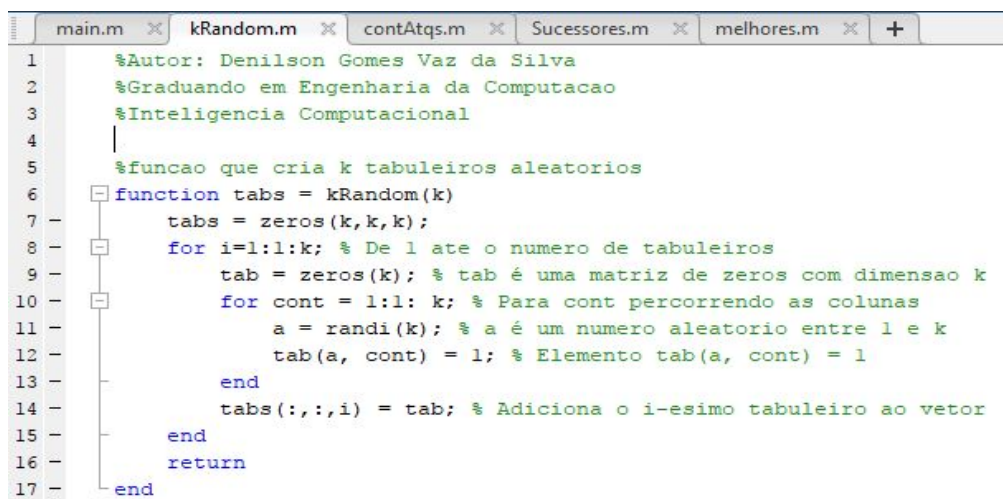
O algoritmo *Brute Force* consiste em sair testando todas as possíveis soluções, já o *Local Beam Search* efetua a busca mantendo k estados e buscando a solução a partir deles.

2. REPRESENTAÇÃO

Para solucionar esse precisamos descrever nossos elementos, o tabuleiro e as rainhas de forma computacional. No caso, decidi tratar o tabuleiro de k linhas por k colunas como uma matriz de k linhas e k colunas, a presença de uma rainha na casa do tabuleiro é representada pelo número '1' na posição, e a ausência de rainha é representado pelo número '0'.

3. IMPLEMENTAÇÃO

Inicialmente foi criado a função "kRandom" que retorna k tabuleiros gerados aleatoriamente, com k rainhas, uma em cada coluna. A função 'kRandom' pode ser vista na figura 1.



```
1 %Autor: Denilson Gomes Vaz da Silva
2 %Graduando em Engenharia da Computacao
3 %Inteligencia Computacional
4
5 %funcao que cria k tabuleiros aleatorios
6 function tabs = kRandom(k)
7     tabs = zeros(k,k,k);
8     for i=1:1:k; % De 1 ate o numero de tabuleiros
9         tab = zeros(k); % tab é uma matriz de zeros com dimensao k
10        for cont = 1:1: k; % Para cont percorrendo as colunas
11            a = randi(k); % a é um numero aleatorio entre 1 e k
12            tab(a, cont) = 1; % Elemento tab(a, cont) = 1
13        end
14        tabs(:,:,i) = tab; % Adiciona o i-esimo tabuleiro ao vetor
15    end
16    return
17 end
```

Figura 1. Função que retorna k tabuleiros aleatórios.

Fonte: Próprio Autor

Em seguida foi criada uma função que retorna o número de ataques de um determinado tabuleiro. A função “contAtqs” foi dividida em três partes.

A primeira parte que gera o vetor linha_Rainha, o qual para cada índice tem o valor da linha em que a rainha da coluna daquele índice se encontra. Essa primeira parte pode ser vista na figura 2.

```
5      %funcao que conta os ataques do tabuleiro
6      function Atqs = contAtqs(tab)
7      [k,~] = size(tab); %k recebe a dimensao do tabuleiro
8      linha_Rainha = zeros(1,k); %linha_Rainha inicia com k zeros
9      Atqs = 0; %Atqs inicia em 0
10
11     %Construir vetor com o valor da linha do indice coluna
12     for c=1:1:k %For p/ colunas
13         for l=1:1:k %For p/linhas
14             %Verifica se a rainha da coluna c esta na linha l
15             if tab(l,c) == 1 %Caso esteja
16                 linha_Rainha(c) = l;
17                 %linha_Rainha na posicao da coluna recebe o valor da linha
18             end
19         end
20     end
21
```

Figura 2. Trecho que cria o vetor com a posição das rainhas.

Fonte: Próprio Autor

Sabendo a posição de todas as k rainhas no tabuleiro já é possível calcular o número de ataques entre elas. Na segunda parte de “contAtqs” contamos os ataques por linha. Essa parte pode ser vista na figura 3.

```
main.m x kRandom.m x contAtqs.m x Sucessores.m x melhores.m x +
21
22     %Calcular ataques por linha
23     for cl=1:1:k-1 %Primeira rainha a ser comparada
24         for c2=cl+1:1:k %Segunda rainha a ser comparada
25             %Compara a linha da cl-esima rainha com todas as rainhas
26             %que ainda nao foi comparada, de cl+1 ate k. De 1 ate cl ja foi comparada.
27             if linha_Rainha(cl) == linha_Rainha(c2)
28                 %Se a linha da cl-esima rainha for igual a linha da
29                 %c2-esima rainha
30                 Atqs = Atqs + 1; %Incrementa o ataque
31             end
32         end %No fim deste for a rainha cl foi comparada com todas as outras rainhas
33     end %No fim deste for contou os ataques por linha
34
```

Figura 3. Trecho que calcula os ataques por linhas.

Fonte: Próprio Autor

Depois de calcular os ataques pelas linhas falta apenas calcular os ataques pelas diagonais, já que não há ataques pelas colunas. Essa terceira parte da função “contAtqs” pode ser vista na figura 4.

```

35      %Contar os ataques por diagonal
36 -   for c1=1:1:k-1 %Percorre ate a penultima coluna
37 -   for c2=c1+1:1:k %Percorre da coluna atual ate a ultima
38       %Verifica se c1 e c2 se atacam por diagonal
39 -       if abs(c1-c2) == abs(linha_Rainha(c1)-linha_Rainha(c2))
40 -           Atqs = Atqs + 1;
41 -           %Caso sim, incrementa Atqs
42 -       end
43 -   end %Final deste for ja calculou os ataques de c1
44 - end %Final deste for ja calculou todos os ataques
45 - return
46 - end

```

Figura 4. Trecho que calcula os ataques por diagonais e retorna o total de ataques. Fonte: Próprio Autor

Foi criado também a função “Sucessores” para gerar todos os sucessores de todos os k tabuleiros. Cada tabuleiro gera $k*(k-1)$ tabuleiros, totalizando $k*k*(k-1)$ sucessores. A função “Sucessores” se divide em duas partes, na primeira parte fazemos com que cada sucessor seja a cópia de seu pai, na segunda parte modificamos uma rainha, criando os sucessores. As duas partes da função Sucessores podem ser vistas nas figuras 5 e 6, respectivamente.

```

5      %funcao que gera os sucessores
6 -   function Sucessores = Sucessores(tabs)
7 -       [k,~,~]=size(tabs); %k recebe a dimensao e quantidade de tabuleiros
8 -       Sucessores = zeros(k,k,k*k*(k-1)); %k tabuleiros geram k*(k-1) Sucessores
9 -       a=1; %Variavel para contar os Sucessores
10
11 -   for t=1:1:k %For para os tabuleiros
12 -       %For para fazer com que cada sucessor seja a copia de seu pai, exceto
13 -       %por uma coluna(uma rainha)
14 -       for i=1:1:k %For para colunas
15 -           for q=1:1:k-1 %For para linhas
16 -               Sucessores(:, :, a) = tabs(:, :, t); %Sucessores recebem o pai
17 -               a = a + 1; %Variavel conta Sucessores
18 -           end
19 -       end
20 -   end
21

```

Figura 5. Função que faz com que os sucessores sejam a cópia de seu pai.

Fonte: Próprio Autor

```

22 - a=1; %a volta a contar o primeiro Sucessor
23 - for i=1:1:k %For para todos os tabuleiros
24 -     %Em cada tabuleiro vou modificar uma rainha de cada vez
25 -     %Colocando a rainha da c-esima coluna em todas as linhas disponiveis
26 -     for c=1:1:k %For para as rainhas (uma em cada coluna)
27 -         for l=1:1:k %For para achar a linha onde a rainha esta
28 -             if tabs(l,c,i) == 1 %Se tiver rainha,
29 -                 linha = l; %Linha onde a rainha estava(para nao colocar mais)
30 -                 break %Para de procurar pela rainha, pois ja achou
31 -             end
32 -             %Tirou a rainha da coluna
33 -         end
34 -
35 -         %For para criar os Sucessores deste tabuleiro modificando a
36 -         %rainha desta coluna, colocando nas linhas disponiveis
37 -         for l=1:1:k %For para as linhas
38 -             if l ~= linha %Se a rainha nao estava nessa linha
39 -                 Sucessores(:,c,a) = 0; %Zera a coluna onde vai reanjar a rainha
40 -                 Sucessores(l,c,a) = 1; %Colocou a rainha em tab(l,c)
41 -                 a = a + 1; %Incrementa o valor a para criar o proximo tab
42 -             end
43 -         end
44 -     end
45 - end
46 - end

```

Figura 6. Trecho da função Sucessores que modifica cada sucessor.

Fonte: Próprio Autor

Depois de gerarmos todos os sucessores de todos os k tabuleiros, verificamos se algum destes e a solução, se for, para a execução, caso a solução não tenha sido gerada, escolhemos os k melhores sucessores. A escolha dos k melhores sucessores e feita com a função “Melhores”. A função “Melhores” pode ser vista na figura 7.

```

main.m x kRandom.m x contAtqs.m x Sucessores.m x melhores.m x +
1 %Autor: Denilson Gomes Vaz da Silva
2 %Graduando em Engenharia da Computacao
3 %Inteligencia Computacional
4
5 %funcao que retorna os melhores Sucessores
6 function melhores = melhores(Sucessores) %Recebe o vetAtqs
7 [k,~,t] = size(Sucessores); %k recebe a dimensao do tabuleiro e t o numero de tabuleiros
8 melhores = zeros(k,k,k); %Inicia melhores como k matrizes de zeros
9 b=1; %Variavel para contar os melhores
10 MAXATQS = 100;
11
12 for a=1:1:MAXATQS %Maximo de ataques (comencando em 1, pois assumimos qua a solucao nao foi gerada)
13     for i=1:1:t %Para todos Sucessores
14         if contAtqs(Sucessores(:,i)) == a %Se tiver a ataques
15             melhores(:,b) = Sucessores(:,i); %Melhor indice b recebe Sucessores indice i
16             if b == k %Se b==k, entao ja temos os melhores
17                 return %Retornamos os melhores sucessores
18             end
19             b = b + 1; %Incrementamos b
20         end
21     end
22 end
23 end

```

Figura 7. Função que seleciona os melhores sucessores.

Fonte: Próprio Autor

Com essas funções já implementadas podemos construir o algoritmo *Local Beam Search*. O algoritmo criado se divide em duas partes, a primeira parte cria os k tabuleiros aleatórios e verifica se algum deles é a solução. A segunda parte executa de fato o *Local Beam Search*. As duas partes deste código podem ser vistas nas figuras 8 e 9, respectivamente.

```

main.m x kRandom.m x contAtqs.m x Sucessores.m x melhores.m x +
1      %Autor: Denilson Gomes Vaz da Silva
2      %Graduando em Engenharia da Computacao
3      %Inteligencia Computacional
4      %Resolucao do problema das k Rainhas nao atacantes em tabuleiro k x k
5
6      clear all %Limpar todas as variaveis
7      clc %Limpar visor
8
9      %Perguntamos o tamanho do tabuleiro
10     k = input('Digite o tamanho do tabuleiro: '); %Atribuimos o tamanho a k
11     tabuleiros = kRandom(k); %Geramos k tabuleiros aleatorios
12
13     %Exibimos os tabuleiros iniciais
14     disp('Tabuleiros iniciais\n');
15     tabuleiros
16
17     %Verificar os k tabuleiros aleatorios
18     for i=1:k %For p/ os tabuleiros
19         if contAtqs(tabuleiros(:,i)) == 0 %Se o tabuleiro nao tiver nenhum ataque
20             disp('Tabuleiro Solucao')
21             tabuleiros(:,i) %Exibe o tabuleiro solucao
22             return %Encerra a execucao
23         end
24     end
25

```

Figura 8. Código que gera e verifica os k -primeiros tabuleiros.

Fonte: Próprio Autor

```

26     %Executar LBS
27     i = 0;
28     max = k*k; %Maximo de iteracoes
29     while i <= max %Enquanto nao extrapolar o numero maximo de iteracoes
30         sucessores = Sucessores(tabuleiros); %Gera sucessores
31         for j=1:k*k*(k-1) %Percorre os sucessores
32             if contAtqs(sucessores(:,j)) == 0 %Se algum sucessor tiver 0 ataques
33                 disp('Tabuleiro Solucao')
34                 sucessores(:,j) %Exibe a solucao
35                 return %Encerra a aplicacao
36             end
37         end
38         tabuleiros = melhores(sucessores); %tabuleiros recebe os melhores sucessores
39         i = i + 1; %incrementa i
40     end
41
42     disp('Melhor Tabuleiro')
43     sucessores(:,j) %Exibe o melhor tabuleiro
44

```

Figura . Código que executa o LBS.

Fonte: Próprio Autor

4. RESULTADOS E COMPARAÇÃO COM BRUTE FORCE

O algoritmo implementado usando Local Beam Search obteve êxito em encontrar a solução. Encontrando rapidamente a solução para tabuleiros pequenos, demorando cada vez mais à medida que os tabuleiros crescem.

A dimensão k do tabuleiro também impacta diretamente no consumo de memória, uma vez que cada tabuleiro é uma matriz com as devidas dimensões (k colunas e k linhas) e cada tabuleiro gera $k \cdot (k-1)$ sucessores, totalizando $k \cdot k \cdot (k-1)$ sucessores.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]Wikipedia, Beam Search.

Disponível em: <https://en.wikipedia.org/wiki/Beam_search>.

Acesso em: 15/03/2018

[2]Department of Software Systems - Tampere University of Technology,
2010. Disponível em: <<http://www.cs.tut.fi/~elomaa/teach/AI2010-4.pdf>>

Acesso em: 15/03/2018