Grafos

Trabalho Prático 2

Heitor Lourenço Werneck
heitorwerneck@hotmail.com

30 de Novembro de 2019

Conteúdo

1 Introdução

O problema das oito rainhas é um problema comumente usado em ciência da computação para demonstrar o conceito de *backtracking*, o problema consiste em colocar n rainhas em um tabuleiro de xadrez $n \times n$ tal que as rainhas não ataquem umas as outras.

Este problema é um caso clássico de um problema de satisfação de condição do campo de inteligência artificial e pesquisa operacional.

Esse problema tem algumas aplicações como por exemplo: controle de tráfego aéreo, teste de VLSI [4], balanceamento de carga [3] e prevenção de *deadlock* [5].

Este trabalho visa solucionar esse problema usando uma modelagem em grafos. Também será explorado o problema de n-rainhas.

O problema foi solucionado usando um algoritmo backtracking para busca de soluções e o algoritmo de Bron-Kerbosch para validação de um estado do tabuleiro.

2 Modelagem e Solução

Seja N o tamanho do tabuleiro (N^2 é o número de quadrados no mesmo).

O grafo G que será utilizado para solucionar o problema das rainhas será não direcionado e não ponderado.

Cada posição do tabuleiro $(i, j) \in \{(a, b) : a \in [0, N-1] \wedge b \in [0, N-1]\}$, que é um quadrado, será mapeado ($TransformaParaVertice : (i, j) \rightarrow \mathbb{N}$) para $i * N + j$, que será um vertice com esse mesmo rótulo.

Logo o grafo $G(V, E)$ tal que V são os vértices do grafo e E as arestas entre os vértices terá a seguinte propriedade: v $\in$ V, sendo $V = \{x \in \mathbb{N} : 0 \leq x \leq N^2 - 1\}$

O mapeamento pode ser visto pelo seguinte tabuleiro 2×2 (tabela 1) modelado como matriz e a figura 1.

É necessário definir quando existirá uma aresta entre dois vértices. Como o problema em grafos que será utilizado para solucionar o problema das rainhas será o clique, logo o modo que as arestas são dispostas é

$Quadrado_{00}$	$Quadrado_{01}$
$Quadrado_{10}$	$Quadrado_{11}$

Tabela 1: Tabuleiro modelado na maneira convencional.

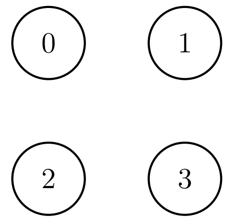

Figura 1: Tabuleiro modelado em grafos.

dependente disso. (Um subconjunto de V($V' \subset V$) será um clique se todos os vértices em V' estão ligados por uma aresta em E)

Então existirá aresta entre dois vértices se não estiverem na mesma linha, coluna ou diagonal.

Essa solução faz sentido pois o clique terá tamanho N em uma solução válida pois todas rainhas irão se conectar ja que não estão na mesma linha, coluna ou diagonal. Logo qualquer clique com tamanho N será uma solução para o problema.

Logo para definir uma regra de criação de arestas será utilizado aritmética modular nos rótulos de cada vértice.

Então dois vértices u' e v' terão seus rótulos dados por u e v respectivamente e assim a equação 2 mostra a condição definida matemáticamente para que seja criado uma aresta.

$$\begin{aligned} \omega(u,v) = (u \bmod N) \neq (v \bmod N) \wedge &\text{ Não estão na mesma coluna} \\ \lfloor u \div N \rfloor \neq \lfloor v \div N \rfloor \wedge &\text{ Não estão na mesma linha} \\ (\lfloor u \div N \rfloor + u \bmod N) \neq (\lfloor v \div N \rfloor + v \bmod N) \wedge &\text{ Não estão na mesma diagonal} \\ (\lfloor u \div N \rfloor - u \bmod N) \neq (\lfloor v \div N \rfloor - v \bmod N) &\text{ Não estão na mesma diagonal} \end{aligned} \tag{2.1}$$

Logo a criação das arestas será dada pelo algoritmo 1.

Algorithm 1 Inserção de arestas.

Input: G

```
procedure EdgePopulator
    for u = 0 to G.|V| - 1 do
        for v = 0 to G.|V| - 1 do
            if ω(u, v) then
                G.E ← G.E ∪ {{u, v}}    ▷ Como o grafo é não direcionado então uma aresta será representada por um conjunto ao invés de uma tupla
```

Com isso a modelagem do grafo está completa e ele pode ser utilizado para solucionar o problema através do clique.

2.1 Algoritmo de clique

O algoritmo de clique utilizado é o Bron-Kerbosch, esse algoritmo acha todos os cliques maximais de um grafo não direcionado. [1]

O algoritmo 2 descreve o comportamento detalhado. A escolha do pivo foi feita pelo vértice com maior grau. A complexidade de tempo do algoritmo tradicional de acordo com a literatura é $O(3^{\frac{n}{3}})$.

Então com o algoritmo de clique definido é possivel verificar a validade de uma combinação de rainhas.

Algorithm 2 Algoritmo de busca de cliques maximais.

Input: G

procedure BRON-KERBOSCH($R, P, X, degrees, N, maximal\_cliques$)
  if $P = \emptyset \wedge X = \emptyset$ **then**
    $maximal\_cliques \leftarrow maximal\_cliques \cup \{R\}$
    return
  $u \leftarrow argmax_{v \in P \cup X} degrees[v]$
  for v in $P \setminus N[u]$ **do**
    Bron-Kerbosch($R \cup \{v\}$,$P \cap N[v]$,$X \cap N[v]$, $degrees, N, maximal\_cliques$)
    $P \leftarrow P \setminus \{v\}$
    $X \leftarrow X \cup \{v\}$
procedure MAXIMUMCLIQUE(G)
  $maximal\_cliques \leftarrow \emptyset$
  Bron-Kerbosch($\emptyset, G.V, \emptyset, G.degrees, G.neighbors, maximal\_cliques$)
  return $max_{c \in maximal\_cliques} |c|$

2.2 Backtracking

Com a base feita pode-se solucionar o problema, basta fazer uma busca pelas soluções. A escolha foi fazer um algoritmo de backtracking tal que tenta colocar rainhas em diversas disposições e se uma disposição não for válida ela não será explorada mais profundamente. O algoritmo 3 descreve detalhadamente o comportamento da solução.

Algorithm 3 Algoritmo de busca das soluções do problema de N rainhas.

Input: G

$solutions \leftarrow \emptyset$
procedure SOLVENQUEENS($column, G, queens$)
  if $column = N$ **then**
    $solutions \leftarrow solutions \cup \{queens\}$
    return
  for $current\_row = 0$ **to** $N - 1$ **do**
    $vertex \leftarrow column + current\_row \cdot N$
    $new\_queens\_set \leftarrow queens \cup \{vertex\}$
    $sub\_graph \leftarrow G.subgraph(new\_queens\_set)$ ▷ Grafo induzido com os vértices do conjunto
    if $MaximumClique(sub\_graph) = column + 1$ **then** ▷ Checa validade do conjunto de rainhas
      SolveNQueens($column + 1, G, new\_queens\_set$)
SolveNQueens($0, G, \emptyset$)

3 Análise de resultados

Os parâmetros que fazem sentido serem variados são o tamanho do tabuleiro juntamente com a quantidade de rainhas.

Primeiro a tabela 2 mostra as soluções obtidas para diversas quantidades de rainhas.

Primeiramente é possivel notar que as soluções são compativeis com a literatura. [2]

Uma parte importante sobre o número de soluções é que algumas são rotações de outras soluções, no caso do problema das 8 rainhas existem 12 soluções únicas. [2]

Outro ponto a se notar é que o problema das rainhas está sendo resolvido com dois algoritmos de complexidade exponencial, o *backtracking* e o algoritmo de clique que faz parte do *backtracking*, isso torna o algoritmo pouco eficiente tanto que não foi possível executar para 12 rainhas.

Na figura 2 é possivel ver que o crescimento do tempo em função do tamanho do número de rainhas é exponencial, comprovando assim a hipótese anterior de que sua complexidade é exponencial.

Rainhas	#Soluções	Tempo(s)
1	1	0.0001
2	0	0.00035
3	0	0.0011
4	2	0.00475
5	10	0.0254
6	4	0.1456
7	40	0.8206
8	92	5.0287500000000005
9	352	32.9279
10	724	208.9649
11	2680	1473.8914

Tabela 2: Resultados.

Pela figura 3 é possível ver o crescimento da quantidade de soluções, entre o problema de 1 e 6 rainhas o crescimento é instável pela própria estrutura do problema, pořem após isso o crescimento é de pelo menos o dobro de soluções.

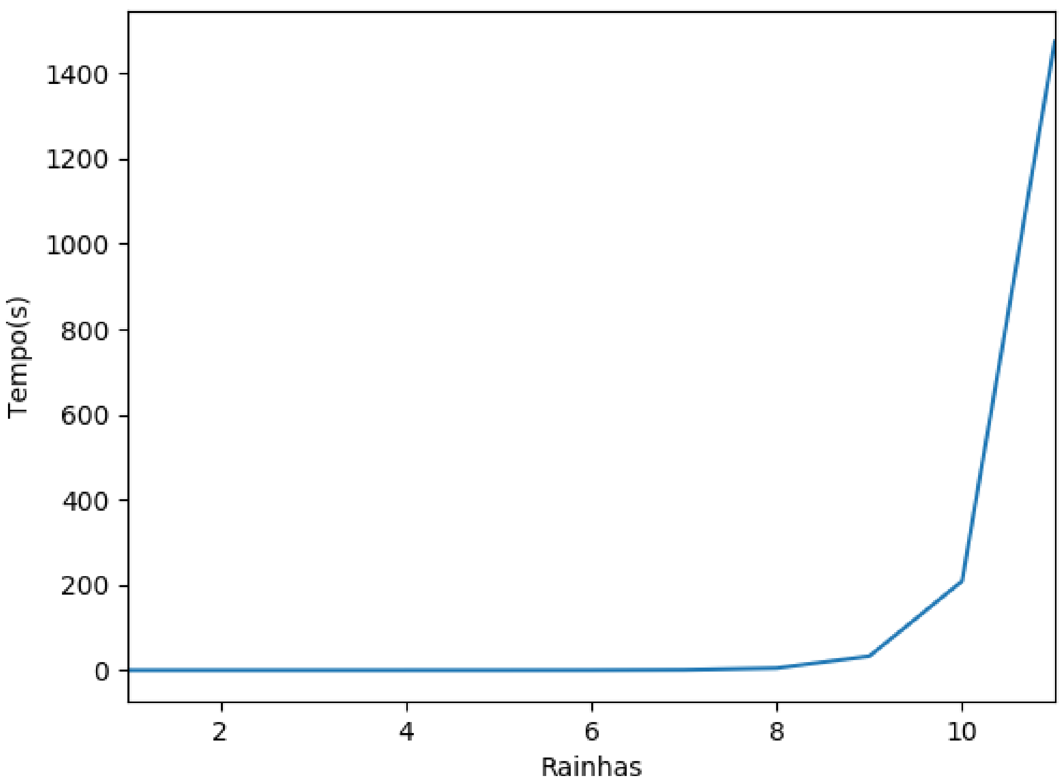

Figura 2: Tempo de execução por número de rainhas.

4 Conclusão

Com esse trabalho foi possível entender que modelando o problema com grafos e resolvendo o mesmo com um algoritmo de um problema NP-Difícil torna o algoritmo muito ineficiente, fora o *overhead* da manipulação das estruturas que representam um grafo.

Porém também foi possível pensar em novas abordagens para se solucionar o problema das 8 rainhas, em trabalhos futuros seria interessante solucionar o mesmo com outros algoritmos de grafos, buscando o

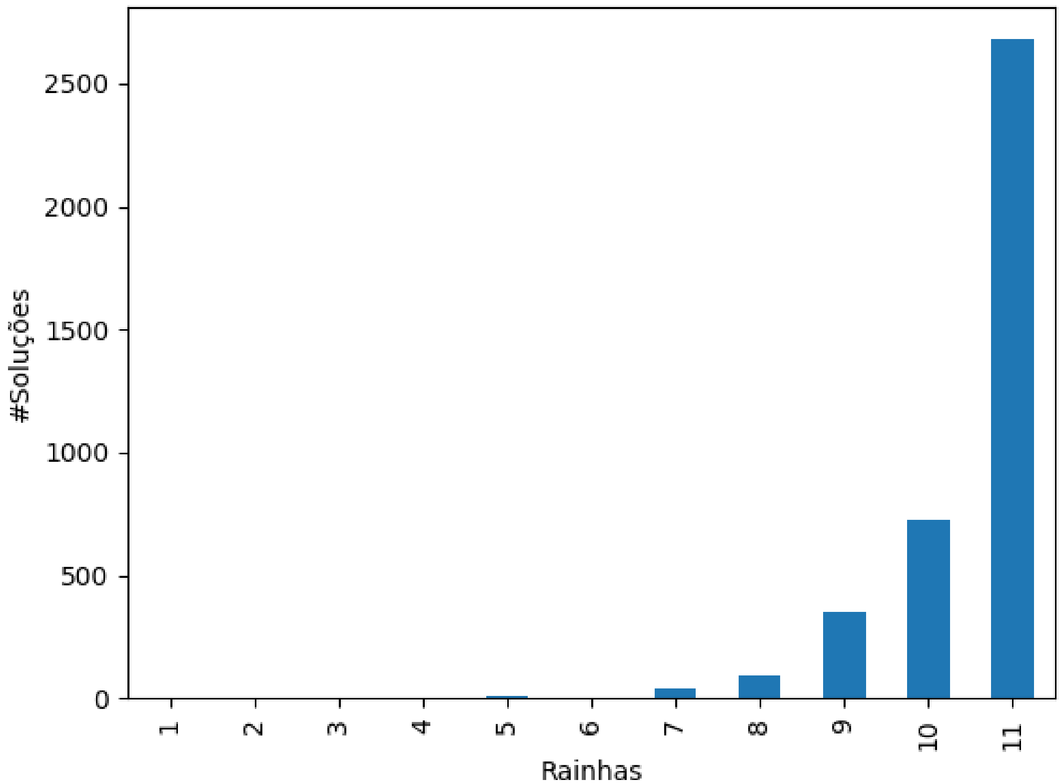

Figura 3: Soluções por número de rainhas.

mais eficiente em termos de complexidade assintótica de tempo.

Os resultados obtidos do número de soluções foram iguais a literatura o que checa a validade da solução em partes.

Referências

[1] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[2] Cengiz Erbas, Seyed Sarkeshik, and Murat M. Tanik. Different perspectives of the n-queens problem. In *Proceedings of the 1992 ACM annual conference on Communications - CSC '92*, page nil, - 1992.

[3] Punita Panwar, Varun Prakash Saxena, Arsch Sharma, and Vijay Kumar Sharma. Load balancing using n-queens problem. 2013.

[4] Rok Sosic and Jun Gu. A polynomial time algorithm for the n-queens problem. *ACM SIGART Bulletin*, 1(3):7–11, 1990.

[5] Murat Mehmet Tanik. A graph model for deadlock prevention. 1978.