UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CENTRO DE CIÊNCIAS DA NATUREZA - CCN
DEPARTAMENTO DE COMPUTAÇÃO

PROBLEMAS NP-COMPLETO
CONJUNTO INDEPENDENTE MÁXIMO

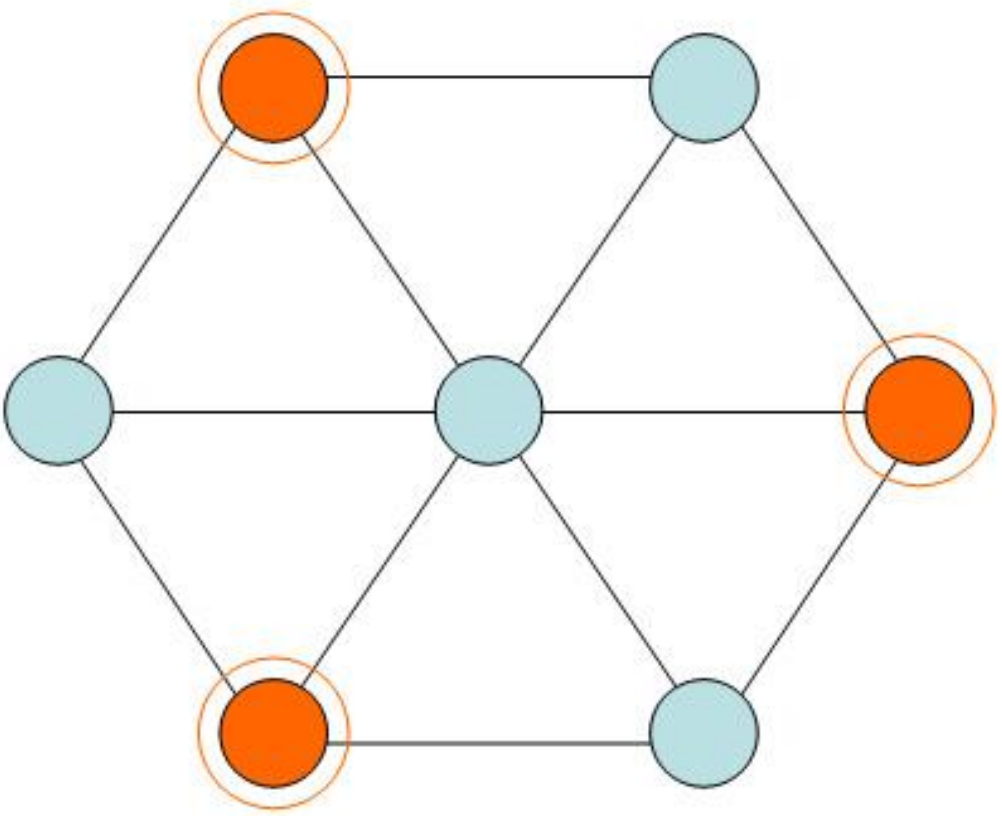
Lucas Bandeira Miranda
Jacob Alexandre Rufino Leoncio Cavalcante de Araújo

Teresina - PI
Julho / 2022

1. Descrição do problema

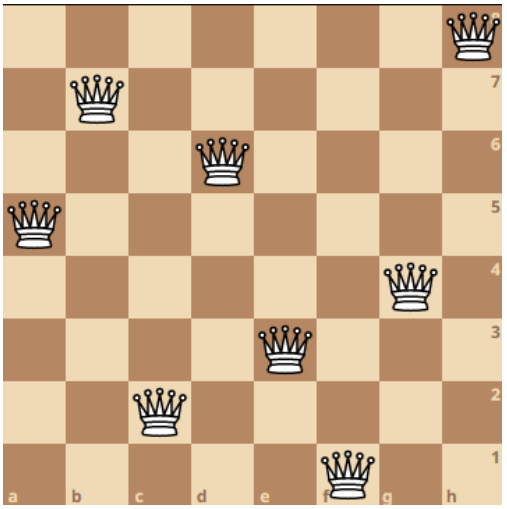
Na teoria dos grafos, um conjunto independente de um grafo **G** é um conjunto **S** de vértices de **G** tal que não existem dois vértices adjacentes contidos em **S**. Em outras palavras, se **a** e **b** são vértices quaisquer de um conjunto independente, não há aresta entre **a** e **b**.

Todo grafo tem ao menos um conjunto independente: o conjunto vazio. Um grafo pode ter vários conjuntos independentes distintos. Além de que em todo grafo não-nulo (grafos que tem pelo menos um vértice) tem um conjunto independente trivial que é formado pela junção dos conjuntos formados por cada vértice único.
Se **S** é um conjunto independente de **G** e não existe um conjunto independente de **G** maior que **S**, diz-se que **S** é um conjunto independente máximo de **G**. O problema de, dado um grafo **G**, determinar se há um conjunto independente de tamanho **k** é um problema NP-completo pois apesar de ser possível achar uma resposta usando algoritmo de força bruta, isto é, procurando dentre todos os subconjuntos dentre o conjunto de vértices do grafo, não é conhecida uma forma de resolver o problema em um tempo satisfatório.



2. Exemplo prático

Para aplicarmos essa teoria de **conjuntos independentes em um exemplo prático**, decidimos utilizar um exemplo de damas em um tabuleiro de xadrez. É possível colocar n damas num tabuleiro de xadrez n-por-n de modo que nenhuma delas possa atacar outra? Nesse exemplo, cada dama representaria um vértice pertencente ao conjunto independente. As arestas do Grafo, seriam representadas por todos os pares de casas que respeitem o movimento da dama do tabuleiro (diagonal, horizontal e vertical), por isso a exigência de que nenhuma dama possa atacar a outra para que assim o conjunto independente seja cumprido. A solução para esse problema é igualmente um problema NP-Completo, pois temos várias vertentes que podemos seguir até a solução ótima (Ou solução aceitável).



Na imagem ao lado observamos um exemplo de arranjo de damas que formam um conjunto independente possível para o nosso tabuleiro de xadrez 8x8.

3. Metodologia utilizada no código de força bruta

A metodologia escolhida para fazer o código de força bruta utilizado neste trabalho é a seguinte: Nosso código percorre todo o grafo criando todos os conjuntos independentes, após isso, separamos os maiores conjuntos independentes (maximais) gerados de tamanho menor ou igual a n (número de casas verticais ou horizontais do tabuleiro), encontrando assim a solução ótima. O código é considerado força bruta pois ele percorre todo o grafo e testa todos os conjuntos independentes até encontrar o(s) maximais.

4. Código força bruta

```python
import networkx as nx
import itertools

numCells = 3

# create a matrix of the board nxn
def create_board(numCells):
    board = []
    for i in range(numCells):
        board.append([])
        for j in range(numCells):
            board[i].append('X')
    return board

def set_board(board, queens):
    for i, j in queens:
        board[i][j] = 'Q'
    return board

# print the board on the chess format
def print_board(board):
    print("Board:")
    for i in range(len(board[0])):
        for j in range(len(board)):
            print(board[j][i], end='  ')
        print('\n', end='')


def clean_board(board):
    for i in range(len(board[0])):
        for j in range(len(board)):
            board[j][i] = 'X'
```

```
33
34  chessBoard = create_board(numCells)
35  print_board(chessBoard)
36
37  # create a graph for represent the chess board n x n
38  cellsBoard = numCells
39  board = nx.Graph()
40  for i in range(cellsBoard):
41      for j in range(cellsBoard):
42          board.add_node((i,j))
43
44  # add edges between the nodes in the horizontal, vertical and horizontal range
45  # this algorithm is not clean because is making a edge between node a and node b and other edge between node b a
46
47  for i in range(cellsBoard):
48      for j in range(cellsBoard):
49          for k in range(cellsBoard):
50              # making the horizontal edges
51              if j != k:
52                  board.add_edge((i, j), (i, k))
53              # making the vertical edges
54              if i != k:
55                  board.add_edge((i, j), (k, j))
56          # making the horizontal range edges
57          # illustration of the diagonals: https://drive.google.com/file/d/1xdbqjWrVbb79z2TbClz5LRUDkT3foabf/view
58          for k in range(1, cellsBoard):
59              if (i - k) >= 0 and (j - k) >= 0:
60                  board.add_edge((i, j), (i - k, j - k))
61              if (i + k) < cellsBoard and (j + k) < cellsBoard:
62                  board.add_edge((i, j), (i + k, j + k))
63              if (i + k) < cellsBoard and (j - k) >= 0:
64                  board.add_edge((i, j), (i + k, j - k))
```

```
            if (i + k) < cellsBoard and (j - k) >= 0:
                board.add_edge((i, j), (i + k, j - k))
            if (i - k) >= 0 and (j + k) < cellsBoard:
                board.add_edge((i, j), (i - k, j + k))

# choose a node and verify if has a edge with the other nodes
candidatesSets = []
queensOnBoard = []
accept = True

for i in range(1, numCells + 1):
    candidatesSets += itertools.combinations(board.nodes(), i)

independentSets = []
approvedSet = True
control1 = 0
for candidate in candidatesSets:
    control1 = 0
    for node1 in range(control1, len(candidate)):
        control2 = 0
        for node2 in range(control2, len(candidate)):
            if candidate[node1] != candidate[node2]:
                if board.has_edge(candidate[node1], candidate[node2]):
                    approvedSet = False
                    break
        if not approvedSet:
            break
        control2 += 1
    if approvedSet:
        independentSets.append(candidate)
    approvedSet = True
    control1 += 1
```

```
93          approvedSet = True
94          control1 += 1
95
96      print('Independent sets: ', independentSets)
97
98      # selecting the maximal sets
99      maximalSets = []
100     maxlength = 0
101     for i in independentSets:
102         if len(i) > maxlength:
103             maximalSets = []
104             maximalSets.append(i)
105             maxlength = len(i)
106         elif len(i) == maxlength:
107             maximalSets.append(i)
108
109     print('Maximal sets: ', maximalSets)
110
111     for sets in maximalSets:
112         clean_board(chessBoard)
113         set_board(chessBoard, sets)
114         print_board(chessBoard)
115         print('Queens: ', sets)
116
117     print(len(maximalSets))
118
119
```

5. Resultados força bruta

O código está conseguindo as soluções ótimas e, como esperado, demorando um tempo inviável para uma solução com um n > 7, já que se trata de um algoritmo força bruta.

Com a execução do código, vimos que nem para todos os casos, o tamanho n por n do tabuleiro vai poder ser igual ao tamanho do conjunto independente máximo. Os casos que o tamanho do conjunto irá divergir do tabuleiro serão para n = 2 e n = 3.

Exemplo de saída para n = 6(tabuleiro 6x6):

```
Board:
X  X  X  Q  X  X
Q  X  X  X  X  X
X  X  X  X  Q  X
X  Q  X  X  X  X
X  X  X  X  X  Q
X  X  Q  X  X  X
Queens:  ((0, 1), (1, 3), (2, 5), (3, 0), (4, 2), (5, 4))
Board:
X  X  X  X  Q  X
X  X  Q  X  X  X
Q  X  X  X  X  X
X  X  X  X  X  Q
X  X  X  Q  X  X
X  Q  X  X  X  X
Queens:  ((0, 2), (1, 5), (2, 1), (3, 4), (4, 0), (5, 3))
Board:
X  Q  X  X  X  X
X  X  X  Q  X  X
X  X  X  X  X  Q
Q  X  X  X  X  X
X  X  Q  X  X  X
X  X  X  X  Q  X
Queens:  ((0, 3), (1, 0), (2, 4), (3, 1), (4, 5), (5, 2))

Board:
X  X  Q  X  X  X
X  X  X  X  X  Q
X  Q  X  X  X  X
X  X  X  X  Q  X
Q  X  X  X  X  X
X  X  X  Q  X  X
Queens:  ((0, 4), (1, 2), (2, 0), (3, 5), (4, 3), (5, 1))
4
```