Métodos de Apoio à Decisão

Problemas de Otimização em Grafos

João Pedro Pedroso

2023/2024

Última aula:

- Método da pesquisa em árvore para problemas específicos
  - problema da mochila
  - problema de sequenciamento de tarefas
- Enumeração implícita

Problemas em grafos

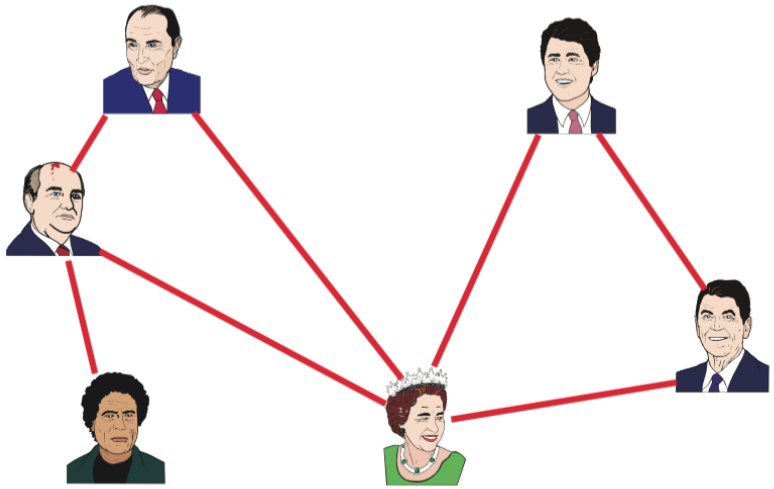
Problemas em grafos

Problemas desta e das próximas aulas:

- Baseados na definição de grafo
  - objeto abstrato composto por vértices e arestas
    - aresta $\rightarrow$ ligação entre dois vértices
  - útil para representar sem ambiguidade muitos problemas reais
  - muitos problemas de otimização podem ser definidos de forma natural através de grafos
- Iremos ver vários problemas de otimização com estrutura combinatória
  - hoje: problema da coloração de grafos

Exemplo: relações de amizade como um grafo

- Temos seis amigos
  - representamos cada um deles por um círculo
  - em teoria de grafos $\rightarrow$ chamados vértices, nodos ou pontos
- Alguns destes amigos estão de boas relações, outros não
- Para organizar estas relações:
  - ligar com uma linha cada par de pessoas em boas relações
  - em teoria de grafos $\rightarrow$ chamadas arestas, arcos ou lines
- Esta representação é um grafo $\rightarrow$ relações de amizade ficam muito fáceis de entender



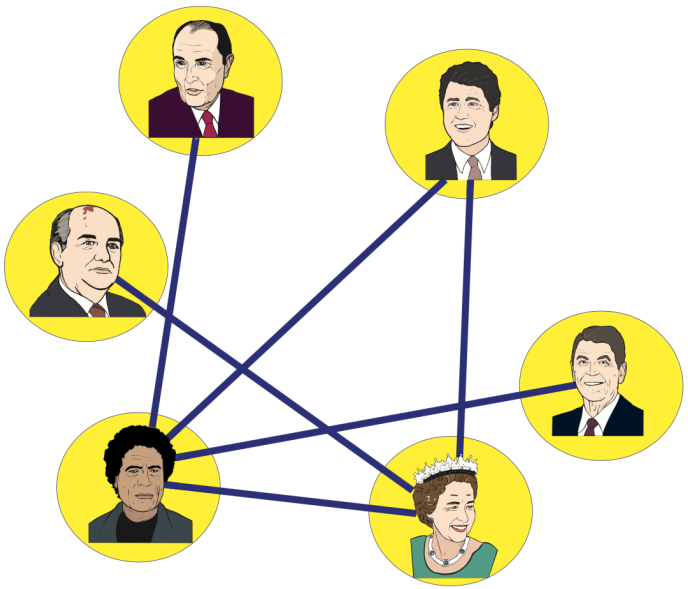
Alguma terminologia

- Grafos não dirigidos: as linhas que ligam dois vértices não têm uma direção implícita
  - ligações sem direção $\rightarrow$ arestas (*edges*)
- Grafos dirigidos: direção nas linhas que ligam dois vértices é importante
  - ligações dirigidas $\rightarrow$ arcos (*arcs*)
- Conjunto de vertices: V
- Conjunto de arestas: E
- Grafo: $G = (V, E)$
- Vértices que são pontos terminais de uma aresta: adjacentes
- Uma aresta é incidente nos vértices que liga
- Número de arestas ligadas a um vértice: grau do vértice

Problema da coloração de grafos

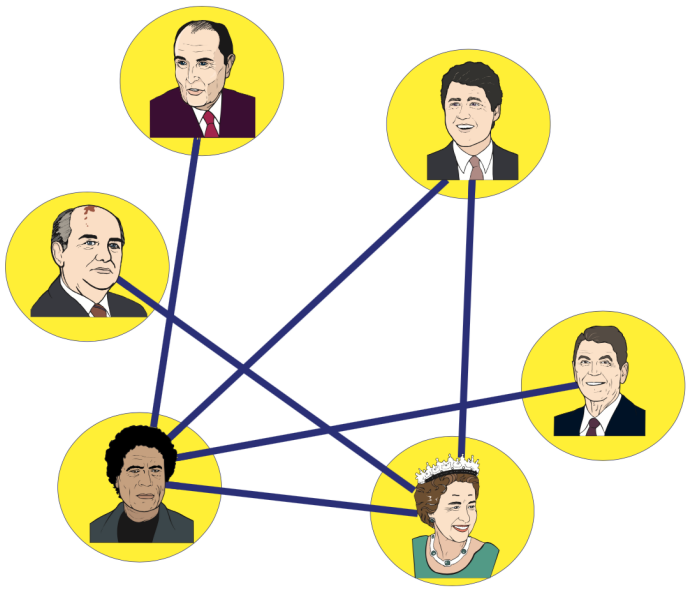
Problema da coloração de grafos

Considere o problema de atribuir uma classe a cada um de seus amigos. Aqueles que estão em relações hostis uns com os outros estão ligados por uma aresta na figura abaixo. Se colocadas na mesma classe, pessoas em más relações começarão uma querela. Como proceder para dividir seus amigos no menor número possível de grupos, de forma que não haja pessoas em relações hostis no mesmo grupo?



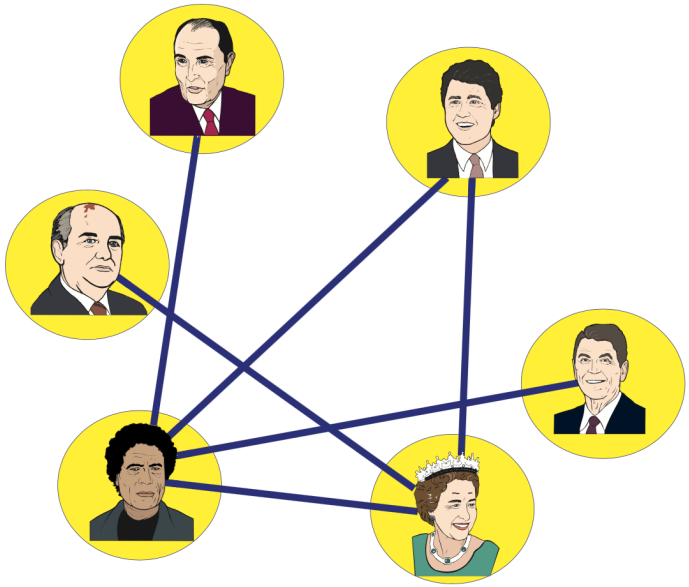
Problema da coloração de grafos

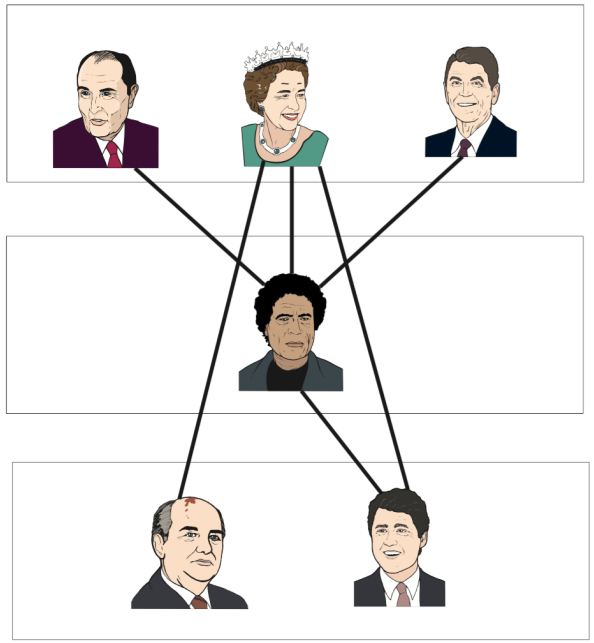
- Grafo:
  - nós $\rightarrow$ pessoas
  - linha entre duas pessoas $\rightarrow$ não são amigos
  - três classes são suficientes para separar as pessoas em más relações



Problema da coloração de grafos

- Grafo:
  - nós $\rightarrow$ pessoas
  - linha entre duas pessoas $\rightarrow$ não são amigos
  - três classes são suficientes para separar as pessoas em más relações





Coloração de grafos: aplicações

1. Coloração de mapas
   - mapas de países
   - países adjacentes não podem ter a mesma cor
   - quatro cores são suficientes ("Teorema das Quatro Cores")
2. Sequenciamento, criação de horários
   - diversos cursos oferecidos
   - vários alunos matriculados em cada curso
   - como agendar exames para que nenhum aluno tenha mais que um exame ao mesmo tempo?
     - vértice $\rightarrow$ curso
     - aresta $\rightarrow$ algum aluno em ambos os cursos
3. Atribuição de frequências de rádio
   - frequências são atribuídas a antenas
   - se as antenas estiverem "suficientemente próximas" $\rightarrow$ interferência $\rightarrow$ frequências devem ser diferentes
     - antena $\rightarrow$ vértice
     - antenas "próximas" $\rightarrow$ unidas por uma aresta

Problema da coloração de grafos

- Dado um grafo não dirigido $G = (V, E)$, uma K partição é uma divisão dos vértices V em K subconjuntos $V_1, V_2, \ldots, V_K$ tal que
  - $V_i \cap V_j = \emptyset, \forall i \neq j \rightarrow$ *não há sobreposição*
  - $\bigcup_{j=1}^{K} V_j = V \rightarrow$ *união = conjunto de todos os vértices*
- Cada $V_i (i = 1, 2, \ldots K)$ denomina-se uma classe de cor.
- Uma K partição tal que em cada classe de cor não há arestas entre vértices dessa class diz-se uma K coloração

Para um dado grafo não dirigido, o problema de coloração de grafos consiste em encontrar K mínimo para o qual existe uma coloração

- isso é chamado de número cromático do gráfico
- aplicações: horários, atribuição de frequências, ...

Notação

- Requisito: limite superior $K_{\max}$ do número de cores
- Número ótimo de cores K: inteiro tal que $1 \leq K \leq K_{max}$
- Variáveis (binárias):
  - $x_{ik} = 1$ se a um vértice i for atribuída a cor k, 0 caso contrário
  - $y_k = 1$ se a cor k foi usada, 0 caso contrário
    - se $y_k = 1 \rightarrow$ conjunto V_k contém pelo menos um vértice
    - se $y_k = 0 \rightarrow V_k$ é vazio *(cor k não foi necessária)*

Formulação matemática

$$\text{minimize} \quad \sum_{k=1}^{K_{\max}} y_k$$

$$\text{subject to} \quad \sum_{k=1}^{K_{\max}} x_{ik} = 1 \qquad \forall i \in V$$

$$x_{ik} + x_{jk} \leq y_k \qquad \forall \{i,j\} \in E; k = 1, \ldots, K_{\max}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in V; k = 1, \ldots, K_{\max}$$

$$y_k \in \{0,1\} \qquad k = 1, \ldots, K_{\max}$$

- Primeira restrição: é atribuída exatamente uma cor a cada vértice
- Segunda restrição: faz a ligação entre as variáveis x e y
  - permite colorir com a cor k somente se $y_k = 1$
  - proíbe as extremidades de qualquer aresta $\{i,j\}$ de terem a mesma cor

Observações

- A maioria dos resolutores (*solvers*) de otimização matemática usa *branch-and-bound*
- Todas as classes de cores na formulação acima são tratadas indiferentemente $\rightarrow$ espaço de soluções tem muita simetria
- A simetria causa problemas ao *branch-and-bound*
  - aumenta enormemente o tamanho da árvore
  - *e.g.*, soluções $V_1 = 1, 2, 3, V2 = 4, 5$ e $V_1 = 4, 5, V2 = 1, 2, 3$ são equivalentes, mas são representadas por diferentes vetores x e y
  - neste caso, ramificar em qualquer uma das variáveis x, y não leva a melhorias no minorante
- Para evitar simetria:
  - $\rightarrow$ usar preferencialmente classes de cores com índices baixos
  - pode melhorar consideravelmente o tempo de resolução

$$y_k \geq y_{k+1} \quad k = 1, \ldots, K_{\max} - 1$$

Modeling tip

Quando houver simetria numa formulação, adicionar restrições para a remover.

- Adicionar restrições para quebrar explicitamente a simetria pode melhorar drasticamente o tempo de solução
- Mas não há diretrizes gerais...
  - não é óbvio decidir que restrições devem ser adicionadas
  - adicionar restrições simples (como atrás) geralmente funciona bem
  - nalguns casos, adicionar restrições elaboradas quebra a estrutura do problema, tornando a reolução mais lenta
- Experimentação cuidadosa é necessária para decidir se tais restrições são úteis

Modelo AMPL

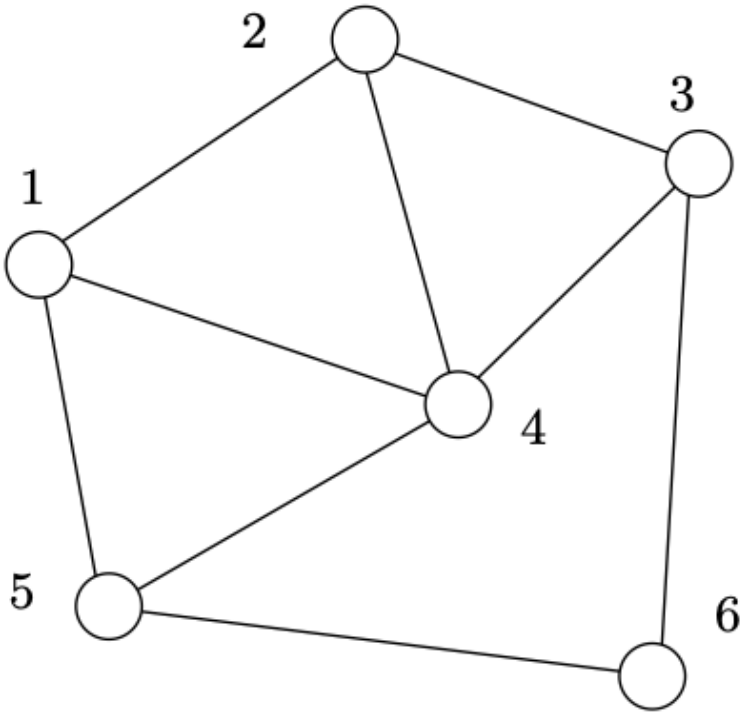
```
set V;                              # vertices
set E within {V,V};                 # edges
param Kmax default card(V);         # upper bound to the number of colors
set C := {1..Kmax};                 # set of colors

var x {V, C} binary;
var y {C} binary;

minimize K: sum {k in C} y[k];

subject to
Color {i in V}: sum {k in C} x[i,k] = 1;
Edge {(i,j) in E, k in C}: x[i,k] + x[j,k] <= y[k];
NoSym {k in 1..Kmax-1}: y[k] >= y[k+1];
```

Como instanciar o grafo?




```
1  set V := 1, 2, 3, 4, 5, 6;
2  set E := (1,2) (1,4) (1,5)
3           (2,3) (2,4) (3,4)
4           (3,6) (4,5) (5,6);
```

Como resolver?

```
model gcp.mod;
data gcp.dat;
option solver gurobi;
option display_1col 0;
solve;
display y;
display x;
```

Output

```
optimal solution; objective 3
53 simplex iterations
y [*] :=
1  1
2  1
3  1
4  0
5  0
6  0
;

x [*,*]
:   1   2   3   4   5   6    :=
1   0   1   0   0   0   0
2   1   0   0   0   0   0
3   0   1   0   0   0   0
4   0   0   1   0   0   0
5   1   0   0   0   0   0
6   0   0   1   0   0   0
;
```

Em Python (ver gcp.py)

```
# V, E = some graph, e.g.:
V = [1, 2, 3, 4, 5, 6]
E = [(1,2), (1,4), (1,5), (2,3), (2,4), (3,4), (3,6), (4,5), (5,6)]

from amplpy import AMPL
ampl = AMPL()
ampl.option['solver'] = 'gurobi'
ampl.read("gcp.mod")
ampl.set['V'] = V
ampl.set['E'] = E
Kmax = len(V)
ampl.param['Kmax'] = Kmax

ampl.solve()
K = ampl.obj['K']
print("Colors used:", K.value())

y = ampl.var['y']
x = ampl.var['x']
for k in range(1,Kmax+1):
    if y[k].value() > .5:
        vk = [i for i in V if x[i,k].value() >.5]
        print(f"color {k} used in {vk}")
```

Coloração de gráfos: nova abordagem

- Modelo anterior: minimizar o número de cores usadas
  - determinar o número cromático K
- Abordagem diferente:
  - número de cores usadas é fixo
  - permite resolver instâncias maiores
- Ideia:
  - corrigir o número de cores
  - não há garantia de que "colorimos" o gráfico
  - "bad edges" têm a mesma cor em ambos os extremos
- Nova variável: $z_{ij} = 1$ se os extremos da aresta $\{i, j\}$ tiverem a mesma cor, 0 caso contrário
- Novo objetivo: minimizar o número de *bad edges*
  - se o ótimo for 0 $\rightarrow$ cores atribuídas são viáveis
    - limite superior para o número cromático K
  - se o ótimo for positivo: há *bad edges*
    - fizar valor $< K$ para o número de cores

Minimizar *bad edges*

$$\text{minimize} \quad \sum_{\{i,j\} \in E} z_{ij}$$

$$\text{subject to} \quad \sum_{k=1}^{K} x_{ik} = 1 \qquad \forall i \in V$$

$$x_{ik} + x_{jk} \leq 1 + z_{ij} \qquad \forall \{i,j\} \in E; k = 1, \ldots, K$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in V; k = 1, \ldots, K$$

$$z_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in E$$

- Primeira restrição: é atribuída exatamente uma cor a cada vértice
- Segunda restrição: arestas $\{i,j\}$ com a mesma cor em i e $j \rightarrow$ *bad edges* $\rightarrow z_{ij} = 1$

AMPL model

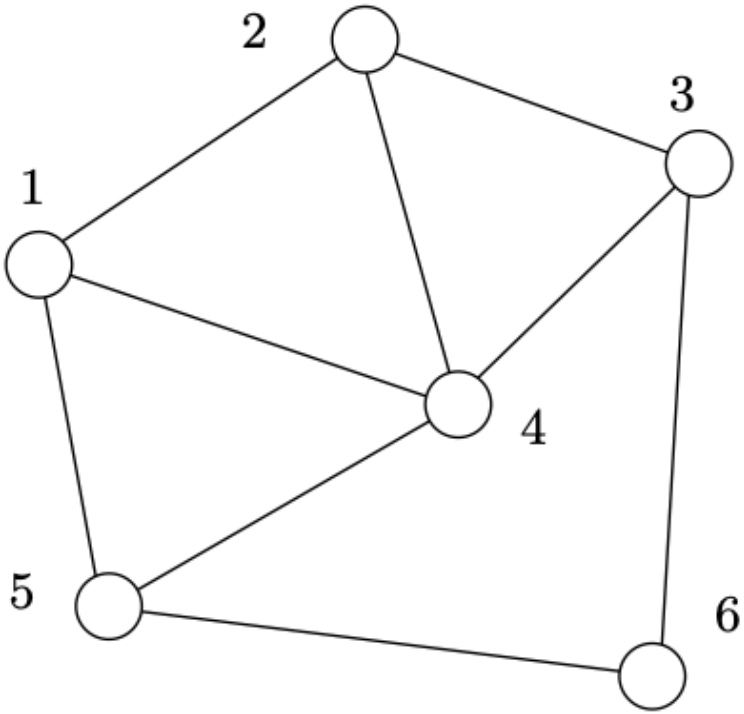
```
set V;
set E within {V,V};
param K;
set C := {1..K};

var x {V, C} binary;
var z {E} binary;

minimize Z: sum {(i,j) in E} z[i,j];

subject to
Color {i in V}: sum {k in C} x[i,k] = 1;
BadEdge {(i,j) in E, k in C}: x[i,k] + x[j,k] <= 1 + z[i,j];
```

Como instanciar?




```
1  set V := 1, 2, 3, 4, 5, 6;
2  set E := (1,2) (1,4) (1,5)
3           (2,3) (2,4) (3,4)
4           (3,6) (4,5) (5,6);
```

Como resolver?

```
model gcp_fixed_k.mod;
data gcp.dat;
option solver gurobi;
option display_1col 0;
solve;
display y;
display x;
```

Output:

```
Z = 2

z [*,*]
:   2   3   4   5   6    :=
1   0   .   1   0   .
2   .   1   0   .   .
3   .   .   0   .   0
4   .   .   .   0   .
5   .   .   .   .   0
;

x [*,*]
:   1   2    :=
1   0   1
2   1   0
3   1   0
4   0   1
5   1   0
6   0   1
;
```

Observações

- K ótimo:
  - menor valor tal que o ótimo para o problemas acima seja 0
  - pode ser determinado através de pesquisa binária
- $\rightarrow$ verificar a melhoria no tempo de CPU usado

Pesquisa binária

Ver `gcp_fixed_k.py`

```
# V, E = some graph, e.g.:
V = [1, 2, 3, 4, 5, 6]
E = [(1,2), (1,4), (1,5), (2,3), (2,4), (3,4), (3,6), (4,5), (5,6)]

from amplpy import AMPL, Environment, DataFrame
ampl = AMPL()
ampl.option['solver'] = 'gurobi'
ampl.read("gcp_fixed_k.mod")
ampl.set['V'] = V
ampl.set['E'] = E
```

```
LB = 0
UB = len(V)
while UB - LB > 1:
    K = (UB + LB) // 2
    print(f"current K:{K}\t[LB:{LB},UB:{UB}]")
    ampl.param['K'] = K
    ampl.solve()
    Z = ampl.obj['Z']
    print("Bad edges:", Z.value())

    if Z.value() > .5:
        LB = K
        z = ampl.var['z']
        for (i, j) in E:
            if z[i, j].value() > .5:
                print("Bad edge:", (i, j))
    else:
        UB = K
        x = ampl.var['x']
        for k in range(1, K + 1):
            vk = [i for i in V if x[i, k].value() > .5]
            print(f"color {k} used in {vk}")

print()
print("Chromatic number:", UB)
```

Nesta aula...

- Problemas em gráfos: coloração
- Modelos:
  - como formular
  - como melhorar a formulação