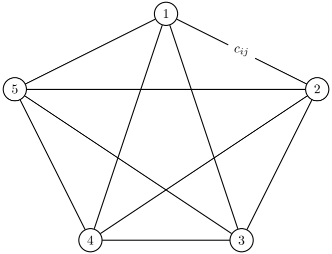
Modelos e algoritmos

Problemas de roteamento

Ricardo Camargo (rcamargo@dep.ufmg.br)

2020

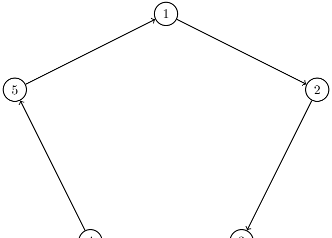
Modelos de programação matemática



Parâmetros

- $N = \{1, \ldots, n\}$
- $A = \{(i, j) : i, j \in N \wedge i \neq j\}$
- $c_{ij} \geq 0$: distância de deslocamento do arco $(i, j) \in A$

Exemplo/solução:

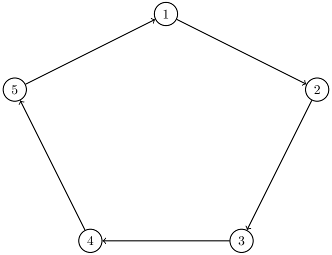


Variáveis de decisão:

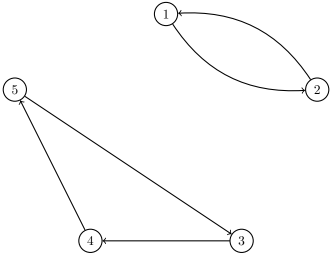
$x_{ij} \in \{0,1\}$: igual a 1 se o arco $(i,j) \in A$ é usado na rota ótima; 0, caso contrário

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} & \\
\text{sujeito a:} \quad & \sum_{(i,j)\in A} x_{ij} = 1 & \forall i \in N \\
& \sum_{(i,j)\in A} x_{ij} = 1 & \forall j \in N \\
& x_{ij} \in \{0,1\} & \forall (i,j) \in A \\
& \text{não existam sub-rotas} &
\end{aligned}
$$

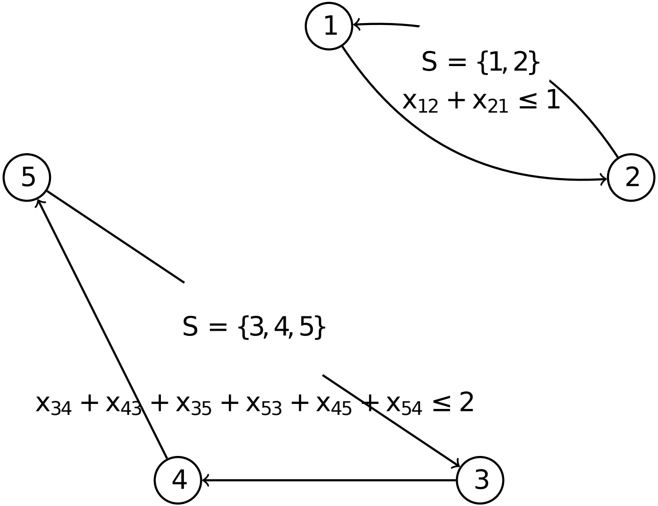
Só com as duas primeiras restrições podemos ter sub-rotas



Uma rota



Duas sub-rotas


1
2
$S = \{1, 2\}$
$x_{12} + x_{21} \leq 1$
5
$S = \{3, 4, 5\}$
$x_{34} + x_{43} + x_{35} + x_{53} + x_{45} + x_{54} \leq 2$
4
3

Restrições de eliminação de sub-rotas:

$$\sum_{(i,j)\in A: i,j\in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset N : |S| \geq 2$$

- S representa subconjuntos de N com mais de 2 e menos de n elementos
- número exponencial de subconjuntos: $2^n - 1$
- S gerado sob demanda, dentro de um algoritmo de plano de cortes

Plano de cortes de DFJ

$stop \leftarrow false$
$\mathcal{S} \leftarrow \{\}$ // Guarda em sub-conjuntos os nós das sub-rotas encontradas
while $stop = false$ **do**
  $\bar{x} \leftarrow DFJ(\mathcal{S})$ // Modelo de DFJ
  $s \leftarrow GetSubTourNodes(\bar{x})$
  if $|s| > 1$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$
  else
    $stop \leftarrow true$
    $\bar{x}$ é a solução ótima
  end if
end while

GetSubTourNodes($\bar{x}$): Como identificar os nós das sub-rotas?

GetSubTourNodes($\bar{x}$): Encontrando os nós das sub-rotas

$\mathcal{V} \leftarrow \{1, \ldots, n\}$ // nós não visitados
$s \leftarrow \{\}$ // conjunto de subconjuntos de nós
while $|\mathcal{V}| > 0$ **do**
    $u \leftarrow \mathit{first}(\mathcal{V})$
    $\mathcal{V} \leftarrow \mathcal{V} \setminus \{u\}$
    $v \leftarrow \mathit{next}(u)$
    $\mathcal{T} \leftarrow \{u\}$
    while $u \neq v$ **do**
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{v\}$
        $\mathcal{V} \leftarrow \mathcal{V} \setminus \{v\}$
        $v \leftarrow \mathit{next}(v)$
    end while
    if $|\mathcal{T}| < n$ **then**
        $s \leftarrow s \cup \{\mathcal{T}\}$
    end if
end while
return s

tspdata.md: comum a todos os modelos

```
param n default 60;

set N := {1..n};
param cx{N} default Uniform(0.0,1000);
param cy{N} default Uniform(0.0,1000);

set A := {(i,j) in {N,N}: i != j};
param c{(i,j) in A} default sqrt( (cx[i] - cx[j])^2 + (cy[i] - cy[j])^2 );
param rt;
param tbbn default 0;
param bbn default 0;
```

dfjtsp.md: modelo DJF

```
param name symbolic := 'dfj';
include tspdata.md;

var x{A}, binary;

minimize of: sum{(i,j) in A} c[i,j] * x[i,j];
s.t. r1{i in N}: sum{(i,j) in A} x[i,j] = 1;
s.t. r2{j in N}: sum{(i,j) in A} x[i,j] = 1;

param nsec default 0;
set H := {1..nsec};
set S{H} within N;
s.t. r3{h in H: card(S[h]) >= 2 and card(S[h]) <= n - 1}: sum{(i,j) in A: i in S[h] and j in S[h]} x[i,j] <= card(S[h]) - 1;

param nxt{N} default -1;
param true := 1;
param false := 0;
param stop default true;
set T within N;
set V within N ordered;
param u;
param v;
param it default 0;
param lb;
end;
```

dfjtsp.run: modelo DJF

```
reset;
model dfjtsp.md;

option seed 0;

option solver cplexamp;
option cplex_options 'lpdisplay = 1 mipdisplay = 2 mipgap = 0.00001 threads = 4';

option solver_msg 0;
option gutter_width 1;
option display_eps 1e-5;
option omit_zero_rows 1;
option omit_zero_cols 1;
```

dfjtsp.run (continuação): modelo DJF

```
problem djf: of, x, r1, r2, r3;

let stop := false;
let rt := time();
repeat while (stop == false)
{
   let it := it + 1;
   solve djf > lixo.txt;
   let bbn := num0(sub(solve_message, '@*^([0-9]+) branch-and@*', '\1'));
   let tbbn := tbbn + if bbn == 0 then 1 else bbn;
   let lb := of;
   if djf.result == 'solved' then
   {
      let{(i,j) in A: x[i,j] > 0.9} nxt[i] := j;
   }
   else
   {
      printf "Failed to solve problem\n";
      exit;
   }
```

dfjtsp.run (continuação): modelo DJF - GetSubTourNodes

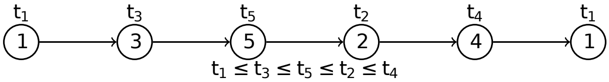
```
    let V := N;
    repeat while (card(V) > 0)
    {
       let u := first(V);
       let V := V diff {u};
       let v := nxt[u];
       let T := {u};
       repeat while (u != v)
       {
           let T := T union {v};
           let V := V diff {v};
           let v := nxt[v];
       }
       if card(T) < n then
       {
          let nsec := nsec + 1;
          let S[nsec] := T;
       }
       else
       {
          let stop := true;
       }
    }
    printf "%3d",it;
    printf " %5d", nsec;
    printf " %10.2f",lb;
    printf "\n";
}
```

dfjtsp.run (continuação): modelo DJF - imprime solução

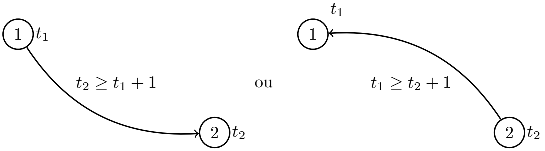
```
printf "Problem  : %12s\n",name;
printf "Cost     : %12.2f\n",lb;
printf "Time     : %12ds\n",time() - rt;
printf "b&b nodes: %12d\n",tbbn;
printf "Route    : ";
printf "%d ", 1;
let v := nxt[1];
repeat while (v != 1)
{
   printf "%d ", v;
   let v := nxt[v];
}
printf "%d ", v;
printf "\n";
end;
```

Variáveis:

- $t_i \geq 0$, ordem ou tempo de visitação do nó $i \in N$



Ou nó 1 é visitado antes do nó 2 ou vice-versa



Restrição de disjunção:

$$t_j \geq t_i + 1 + n(x_{ij} - 1) \qquad \forall (i,j) \in A : j \neq 1$$

Implicações:

$$x_{ij} = 1 \longrightarrow t_j \geq t_i + 1$$

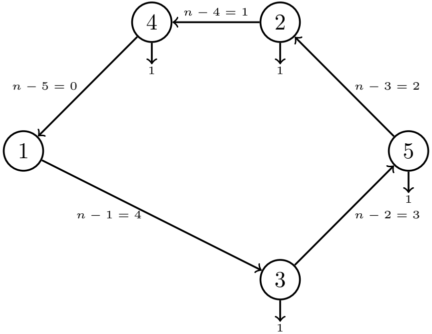
$$x_{ij} = 0 \longrightarrow t_j \geq t_i - n$$

Neste caso, como $t_i \geq 0$ e no máximo n, então $t_i - n$ será um valor negativo ou 0. Então, quando $x_{ij} = 0$, dizemos que a restrição $t_j \geq t_i + 1$ foi "*desativada*" ou "*desabilitada*".

$$
\begin{aligned}
\min \sum_{(i,j)\in A} c_{ij}x_{ij} & & \\
\text{sujeito a: } \sum_{(i,j)\in A} x_{ij} = 1 & & \forall i \in N \\
\sum_{(i,j)\in A} x_{ij} = 1 & & \forall j \in N \\
x_{ij} \in \{0,1\} & & \forall (i,j) \in A \\
t_j \geq t_i + 1 + n(x_{ij} - 1) & & \forall (i,j) \in A : j \neq 1 \\
t_i \geq 0 & & \forall i \in N
\end{aligned}
$$

Ideia:

Cria-se uma quantidade $n - 1$ de fluxo fictício que deve sair da origem e ser entregue uma unidade em cada nó.



Variáveis:

$f_{ij} \geq 0$: quantidade de fluxo fictício passando pelo arco $(i, j) \in A$

Restrições de balanço de fluxo:

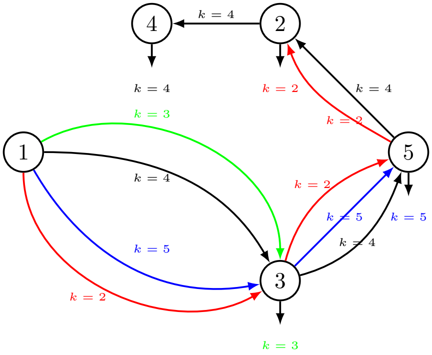
$$
\begin{aligned}
&\sum_{(1,j)\in A} f_{1j} = n - 1 && \\
&\sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = 1 && \forall j \in N \setminus \{1\} \\
&f_{ij} \leq (n-1)x_{ij} && \forall (i,j) \in A \\
&f_{ij} \geq 0 && \forall (i,j) \in A
\end{aligned}
$$

Formulação Gavish and Gaves (1978) ou single-commodity ou mono-produto:

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} & \\
\text{sujeito a:} \quad & \sum_{(i,j)\in A} x_{ij} = 1 & \forall i \in N \\
& \sum_{(i,j)\in A} x_{ij} = 1 & \forall j \in N \\
& x_{ij} \in \{0, 1\} & \forall (i,j) \in A \\
& \sum_{(1,j)\in A} f_{1j} = n - 1 & \\
& \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = 1 & \forall j \in N \setminus \{1\} \\
& f_{ij} \leq (n-1) x_{ij} & \forall (i,j) \in A \\
& f_{ij} \geq 0 & \forall (i,j) \in A
\end{aligned}
$$

Ideia:

Indexar cada unidade de fluxo fictício a ser entregue na própria variável de fluxo. Um caminho para cada produto é formado.



Variáveis:

$f_{ij}^k \geq 0$: porcentagem de fluxo com destino ao nó $k \in N \setminus \{1\}$ que passa no arco $(i,j) \in A$.

Restrições de balanço de fluxo:

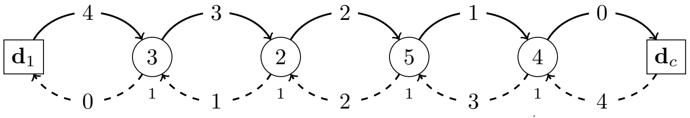
$$
\begin{aligned}
&\sum_{(1,j)\in A} f_{1j}^k = 1 && \forall k \in N \setminus \{1\} \\
&\sum_{(i,j)\in A: i \neq k} f_{ij}^k = \sum_{(j,i)\in A} f_{ji}^k && \forall k, j \in N \setminus \{1\} : k \neq j \\
&\sum_{(i,k)\in A} f_{ik}^k = 1 && \forall k \in N \setminus \{1\} \\
&f_{ij}^k \leq x_{ij} && \forall (i,j) \in A, k \in N \setminus \{1\} \\
&f_{ij}^k \geq 0 && \forall (i,j) \in A, k \in N \setminus \{1\}
\end{aligned}
$$

Formulação Claus (1984) ou multi-commodity ou multi-produto:

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} & \\
\text{sujeito a:} \quad & \sum_{(i,j)\in A} x_{ij} = 1 & \forall i \in N \\
& \sum_{(i,j)\in A} x_{ij} = 1 & \forall j \in N \\
& x_{ij} \in \{0,1\} & \forall (i,j) \in A \\
& \sum_{(1,j)\in A} f^k_{1j} = 1 & \forall k \in N \setminus \{1\} \\
& \sum_{(i,j)\in A : i \neq k} f^k_{ij} = \sum_{(j,i)\in A} f^k_{ji} & \forall k, j \in N \setminus \{1\} : k \neq j \\
& \sum_{(i,k)\in A} f^k_{ik} = 1 & \forall k \in N \setminus \{1\} \\
& f^k_{ij} \leq x_{ij} & \forall (i,j) \in A, k \in N \setminus \{1\} \\
& f^k_{ij} \geq 0 & \forall (i,j) \in A, k \in N \setminus \{1\}
\end{aligned}
$$

Ideia:

Usar dois tipos de fluxos fictícios, quantidade a ser entregue e capacidade residual de retorno, para fazer o balanço de fluxo nos nós.



Variáveis:

$f_{ij} \geq 0$: quantidade de fluxo passando pelo arco $(i, j) \in A$, ou a sua capacidade residual.

Restrições de balanço de fluxo:

$$\sum_{(1,j) \in A} f_{1j} = n - 1$$

$$\sum_{(i,1) \in A} f_{i1} = 0$$

$$\sum_{(i,1^c) \in A} f_{i1^c} = 0$$

$$\sum_{(1^c,j) \in A} f_{1^c j} = n - 1$$

$$\sum_{(i,j) \in A} f_{ij} - \sum_{(j,i) \in A} f_{ji} = 2 \qquad \forall j \in N \setminus \{1, 1^c\}$$

$$\sum_{(i,j) \in A} f_{ij} + \sum_{(j,i) \in A} f_{ji} = 2(n - 1) \qquad \forall j \in N \setminus \{1, 1^c\}$$

$$f_{ij} + f_{ji} = (n - 1)(x_{ij} + x_{ji}) \qquad \forall (i, j) \in A : i < j$$

Formulação Finke, Claus, Gunn (1984) ou two-commodity ou dois produtos:

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} & \\
\text{sujeito a:} \quad & \sum_{(i,j)\in A} x_{ij} = 1 & \forall i \in N \\
& \sum_{(i,j)\in A} x_{ij} = 1 & \forall j \in N \\
& x_{ij} \in \{0,1\} & \forall (i,j) \in A \\
& \sum_{(1,j)\in A} f_{1j} = n-1 & \\
& \sum_{(i,1)\in A} f_{i1} = 0 & \\
& \sum_{(i,1^c)\in A} f_{i1^c} = 0 & \\
& \sum_{(1^c,j)\in A} f_{1^cj} = n-1 & \\
& \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = 2 & \forall j \in N \setminus \{1,1^c\} \\
& \sum_{(i,j)\in A} f_{ij} + \sum_{(j,i)\in A} f_{ji} = 2(n-1) & \forall j \in N \setminus \{1,1^c\} \\
& f_{ij} + f_{ji} = (n-1)(x_{ij}+x_{ji}) & \forall (i,j) \in A : i<j \\
& f_{ij} \geq 0 & \forall (i,j) \in A
\end{aligned}
$$

Atividades a serem feitas

- Implementar os modelos em ampl:
  - Miller, Tucker e Zemlin (1960)
  - Claus (1984) ou multi-commodity ou multi-produto
  - Finke, Claus, Gunn (1984) ou two-commodity ou dois produtos
  - Combinar DFJ com:
    - com a MTZ
    - om a FCG (1984).
    - obs: Iterar por $h = \frac{n}{10}$ iterações só com DFJ, antes de usar as SECs nos modelos MTZ e FCG
- executar todos os modelos apresentados para valores de $n = \{10, 20, 30, \ldots, 100, 150, \ldots, 300\}$, dando um tempo limite de 30 minutos para cada execução. Reportar, em forma de tabela, todos os valores obtidos para o gap final obtido em %, tempo de execução gasto, número de nós de branch-and-bounds usados, número de iterações para o caso específico do modelo DFJ. (Dica: procurar na web como fazer o ampl parar por tempo, e como retornar o mip gap–diretiva return mipgap)