Métodos de Apoio à Decisão

Problemas de rotas, Cutting plane algorithm

João Pedro Pedroso

2023/2024

- Última aula:
  - problema do caixeiro viajante (*traveling salesman problem*)
  - problema do caixeiro viajante assimétrico (*asymmetric traveling salesman problem*)
- Esta aula:
  - problema de rotas de veículos (*vehicle routing problems*)
  - algoritmo dos planos de corte de Gomory

Traveling salesman problem (TSP): definition

Given:

- *undirected graph $G = (V, E)$ with n vertices (cities)*
- *distances $c_e, \forall e \in E$*

Find a tour which passes exactly once in each city and minimizes the total distance (i.e., the length of the tour)

- symmetric case
- Dantzig-Fulkerson-Johnson's (DFJ's) model
- Variables: $x_e \rightarrow$ edges selected for the tour:
  - $x_e = 1$ if edge $e \in E$ is in the tour, 0 otherwise

Mathematical model

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e \\
\text{subject to} \quad & \sum_{e \in \delta(\{i\})} x_e = 2 && \forall i \in V \\
& \sum_{e \in E(S)} x_e \leq |S| - 1 && \forall S \subset V, 2 \leq |S| \leq |V| - 2 \\
& x_e \in \{0, 1\} && \forall e \in E
\end{aligned}
$$

- How to solve it?

How to solve it

Let us start with a simplified version of file `tsp.mod`, only with *degree constraints*:

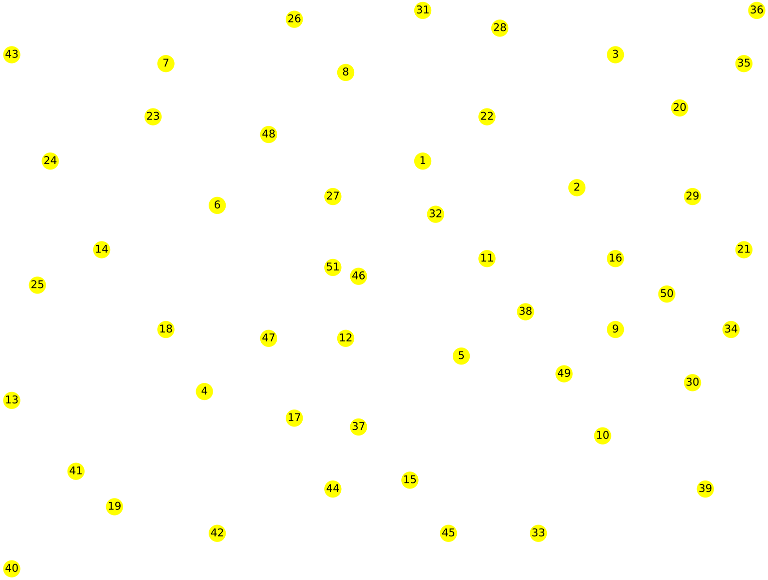
```
set V;
param c {i in V, j in V: i < j};
var x {V, V} binary;

minimize z: sum {i in V, j in V: i < j} c[i,j] * x[i,j];

subject to
Degree {i in V}:
    sum {j in V: j < i} x[j,i] + sum {j in V: j > i} x[i,j] = 2;
```

- we are discarding *subtour elimination constraints*
- initially, variables are continuous

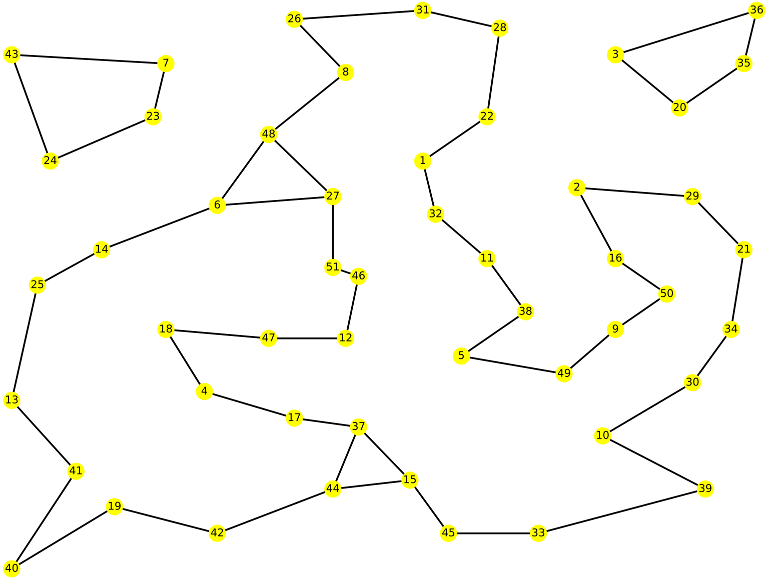
```
ampl.option['relax_integrality'] = True
```

- what does the solution look like?

Instance eil51, just the vertices/cities



Instance eil51, initial solution (no subtour elimination)



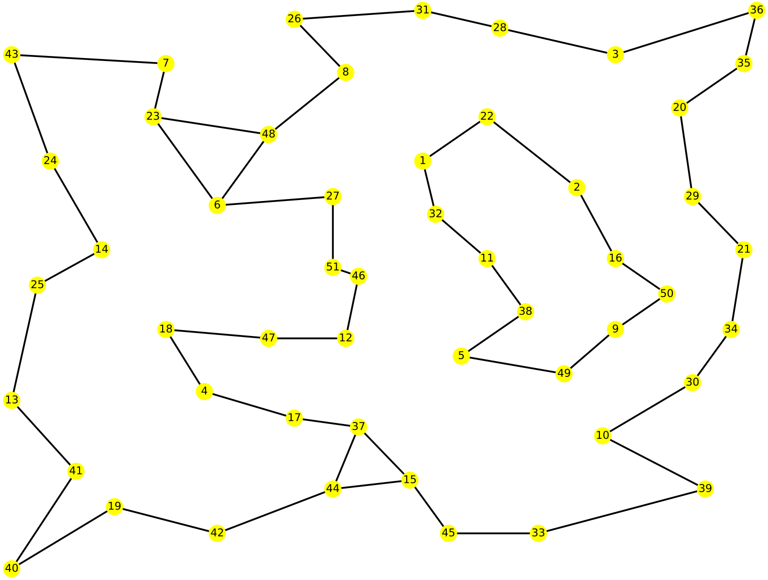
- We must eliminate subtours; *e.g.*, `[7, 23, 24, 43]`
- DFJ's original paper: check flow from vertex 1 to any other vertices
  - max flow $< 2 \quad \Rightarrow$
    - check vertices on corresponding *minimum cut problem*, $(S, V \setminus S)$
    - force $\sum_{x_{\{ij\}}: i \in S, j \in V \setminus S} \geq 2$
- Our simplified version: check *connected components*
  - `[7, 23, 24, 43]` + `[3, 20, 35, 36]` + all other vertices
- Constraints to add:
  - $S = \{7, 23, 24, 43\} \rightarrow x_{7,23} + x_{7,24} + x_{7,43} + x_{23,24} + x_{23,43} + x_{24,43} \leq 3$
  - $S = \{3, 20, 35, 36\} \rightarrow x_{3,20} + x_{3,35} + x_{3,36} + x_{20,35} + x_{20,36} + x_{35,36} \leq 3$
  - $S = \{\text{remaining vertices}\}$
    (long expression)

Back to our AMPL model:

```
param n;    # number of cutting planes
set S {1..n} within V;

Sep {k in 1..n}:
    sum {i in S[k], j in S[k]: i < j} x[i,j] <= card(S[k]) - 1;
```

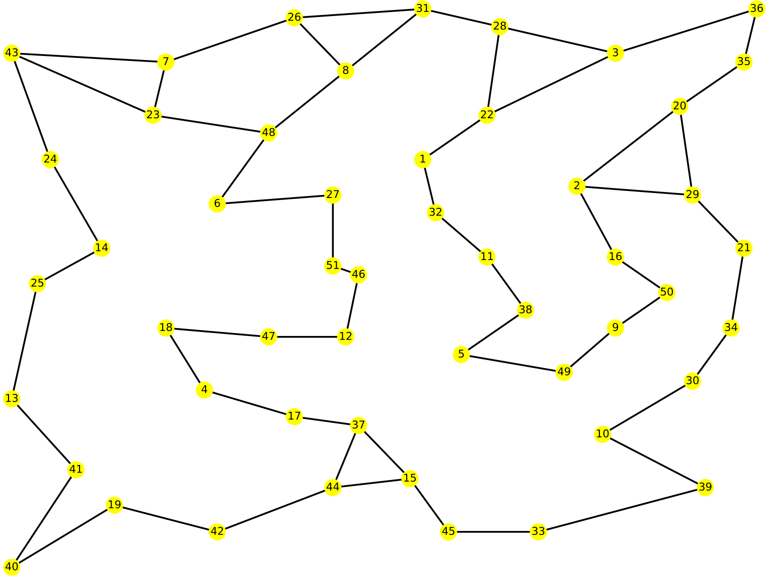
- $k = 1 \rightarrow S = \{7, 23, 24, 43\}$
- $k = 2 \rightarrow S = \{3, 20, 35, 36\}$
- ...

Next iteration: (3 subtour elimination constraints)



- two connectect components:
  - `[1, 22, 2, 16, 50, 9, 49, 5, 38, 11, 32]`
  - remaining vertices

Next iteration: (5 subtour elimination constraints)

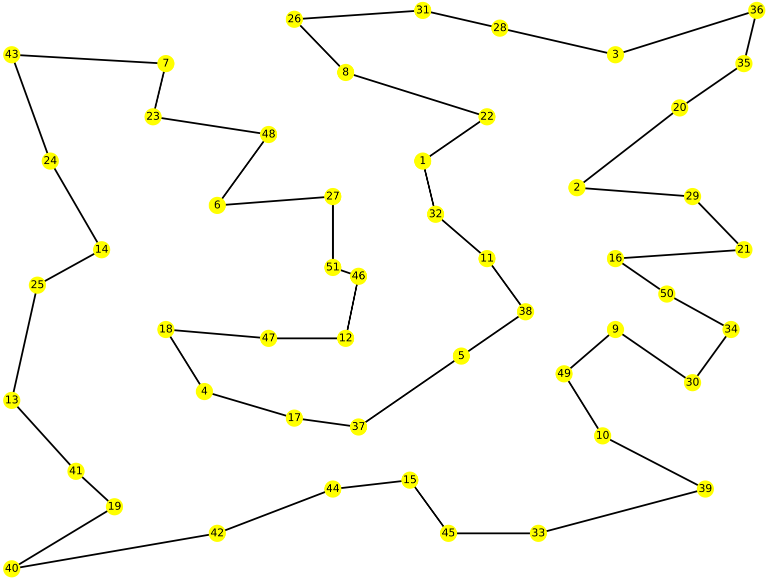


- one connected component
- solution is not integer; we could:
  1. find other inequalities to make the formulation stronger, keeping linear programming model
  2. make variables integer

```
ampl.option['relax_integrality'] = True
```

- doing the latter:

Next iteration: (5 subtour elimination + integer variables)



Notes:

- Check programs ampl-tsp.py tsp.mod and ampl-tsp-plot.py
- Python modules required: `amplpy`, `networkx`, `matplotlib` (for plotting)

Summary:

- Some problems are difficult to formulate in a concise manner
- Number of constraints may be extremely large
- Solution:
    - solve relaxed problem (with only small subset of constraints)
    - add only violated constraints in current solution
    - iterate, until solution is feasible

Asymmetric TSP

Miller-Tucker-Zemlin (potential) formulation

- Consider an asymmetric traveling salesman problem
- We will now see a formulation with a number of constraints of polynomial order

  Given:
  - *directed graph* $G = (V, A)$, *where*
    - V → *set of vertices*
    - A → *a set of (directed) arcs*
  - *function* $c : A \to \mathbb{R}$ *associating a distance to each arc*

  Find a tour which passes exactly once in each city and minimizes the total distance

- Variables:
  - (binary) $x_{ij} = 1$ if visiting vertex j next to i, 0 otherwise
  - (real) u_i → represents the visiting order of vertex i

MTZ formulation for the asymmetric traveling salesman problem

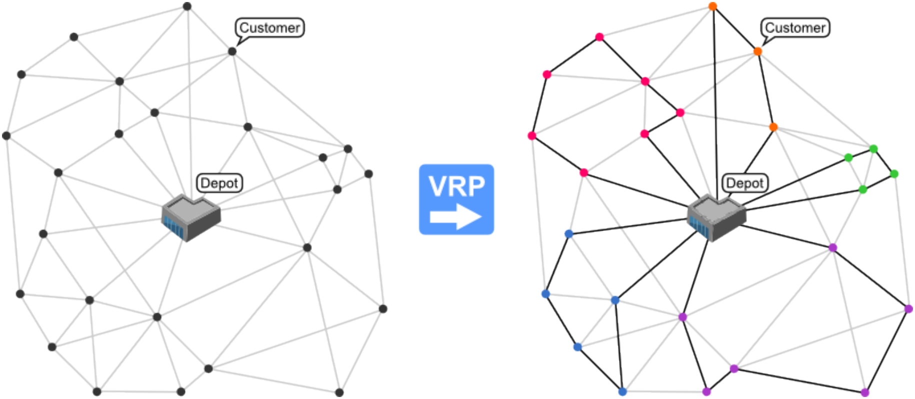
$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \neq j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j: j \neq i} x_{ij} = 1 && i = 1, \ldots, n \\
& \sum_{j: j \neq i} x_{ji} = 1 && i = 1, \ldots, n \\
& u_i + 1 - (n-1)(1 - x_{ij}) \leq u_j && i = 1, \ldots, n, \\
& && j = 2, \ldots, n : i \neq j \\
& 0 \leq u_i \leq n - 1 && i = 1, \ldots, n \\
& x_{ij} \in \{0, 1\} && \forall i \neq j
\end{aligned}
$$

MTZ formulation

- Interpretation: u_i is the potential at each vertex i
  - starting point: $u_1 = 0$
- When visiting j next to vertex i:
  - force the potential of j to be $u_j = u_i + 1$
  - do this for any vertex except 1
  - possible values for u_i are $1, 2, \ldots ..., n - 1$
    - $n \rightarrow$ number or vertices

Capacitated Vehicle Routing Problem

- Practical extension of the TSP; some assumption:
- Depot: point from which all the vehicles depart
  - all vehicles start at the depot, visit some customers, and return to the depot
  - route: order of visit of customers by a vehicle
- Capacity: maximum load that can be carried by each vehicle
- Each customer's location and demand are known;
  - A customer's demand amount does not exceed the vehicles' capacity
  - Each customer is visited exactly once
- Cost of moving between points is known $\rightarrow$ *distance*
- Total amount of customers' demand in one route cannot exceed the vehicle's capacity $\rightarrow$ *capacity constraint*
- The number of vehicles is predetermined, and are all identical


Customer
Depot
VRP
Customer
Depot

Notation

- $m \rightarrow$ number of vehicles
- $n \rightarrow$ number of vertices (customers and depots)
- $q_i \rightarrow$ demand of customer i
- $Q \rightarrow$ capacity of vehicles $k = 1, \ldots, m$
- $c_{ij} \rightarrow$ cost of moving from i to j (assumed symmetric)
- objective: find route of the m vehicles that satisfies all customers' demand at minimum cost
- variables: $x_{ij} \rightarrow$ number of times the vehicle moves between i and j
  - symmetric problem $\rightarrow x_{ij}$ defined only for $i < j$
  - $x_{ij} = 1$ if vehicle takes edge (i, j), 0 otherwise $\rightarrow$ *for all vertices except the depot*
  - $x_{1j} = 0, 1,$ or 2 $\rightarrow$ for the depot
    - *e.g.*, a vehicle may leave the depot, serve customer j, and return $\rightarrow x_{1j} = 2$

Formulation

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i,j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j} x_{1j} = 2m \\
& \sum_{j} x_{ij} = 2 && \forall i = 2, \ldots, n \\
& \sum_{i,j \in S} x_{ij} \leq |S| - N(S) && \forall S \subset \{2, \ldots, n\}, |S| \geq 3 \\
& x_{1j} \in \{0, 1, 2\} && \forall j = 2, \ldots, n \\
& x_{ij} \in \{0, 1\} && \forall i < j : i \neq 1
\end{aligned}
$$

Remarks

- First constraint: m vehicles depart from the depot (vertex 1)
  - number of edges entering and leaving vertex 1 is $2m$
- Second constraint: a single vehicle visits each customer
- Third constraint:
  - capacity constraints of the vehicles
  - prohibits partial routes (subtours)
  - $N(S)$: function of $S \rightarrow$ customer's subset
    $N(S) =$ number of vehicles required to carry demand within S
    - exact method: solve a bin packing problem
    - in general the following lower bounds are used:
      $$\lceil \sum_{i \in S} q_i / Q \rceil$$
- Solution: branch-and-cut as in the TSP
  - check connected components on graph defined by positive $\bar{x}_e, e \in E$
  - add cutting plane

Infeasible instances

- It may happen that the vehicles have no capacity for serving all the demand
- In that case, the problem is infeasible
- How to detect it in AMPL?
  → `ampl.getValue('solve_result')`

string	interpretation
`solved`	optimal solution found
`solved?`	optimal solution indicated, but error likely
`infeasible`	constraints cannot be satisfied
`unbounded`	objective can be improved without limit
`limit`	stopped by a limit that you set (such as on iterations)
`failure`	stopped by an error condition in the solver routines

AMPL model

```
param m;
param n;
set V := {1..n};
param c {i in V, j in V};
var x {i in V, j in V: i<j} integer,
      >= 0, <= (if i == 1 then 2 else 1);
minimize z: sum {i in V, j in V: i < j} c[i,j] * x[i,j];
subject to
Depot:  sum {j in 2..n} x[1,j] = 2*m;
Degree {i in 2..n}:  sum {j in 1..i-1} x[j,i] +
                     sum {j in i+1..n} x[i,j] = 2;
param p;   # number of cutting planes
set S {1..p} within V;
param N {1..p};
Sep {k in 1..p}:
    sum {i in S[k], j in S[k]: i < j} x[i,j] <=
                             card(S[k]) - N[k];
```

Implementation: finding connected components in current solution

```
def addcut(ampl, edges, q, Q):
    G = networkx.Graph()
    G.add_edges_from(edges)
    Components = list(networkx.connected_components(G))
    if len(Components) == 1:
        return False
    cut = False
    p = int(ampl.param['p'].value() + .5)    # number of cuts added
    for S in Components:
        S_card = len(S)
        q_sum = sum(q[i] for i in S)
        NS = int(math.ceil(float(q_sum)/Q))
        S_edges = [(i,j) for i in S for j in S if i<j and (i,j) in edges]
        if S_card >= 3 and (len(S_edges) >= S_card or NS > 1):
            print("adding cut; S={}, |S|={}, N(S)={}".format(S, S_card, NS))
            p += 1
            ampl.param['p'] = p
            ampl.set['S'][p] = S
            ampl.param['N'][p] = NS
            cut = True
    return cut
```

Implementation: solving the VRP

```
def solve_vrp(n, c, m, q, Q):
    V = list(range(1,n+1))
    from amplpy import AMPL, Environment, DataFrame
    ampl = AMPL()
    ampl.read("vrp.mod")
    ampl.param['m'] = m
    ampl.param['n'] = n
    ampl.param['c'] = c
    ampl.param['p'] = 0    # number of cuts
    while True:
        ampl.solve()
        stat = ampl.getValue('solve_result')
        if stat != "solved":
            print("problem is infeasible")
            exit(0)
        z = ampl.obj['z']
        x = ampl.var['x']
        edges = [(i,j) for i in V for j in V if i < j and x[i,j].value() > EPS]
        print("current obj:", z.value(), stat)
        print("edges:", edges)
        if addcut(ampl, edges, q, Q) == False:
            break
    return z.value(), edges
```

Summary

- For several routing problems we have:
- Shown formulations and methods for solving them
    - cutting plane method
- Described approaches for computational solution
    - AMPL models
    - Python programs

Cutting plane algorithm for general integer optimization

Algoritmo dos planos de corte

- problemas lineares com variáveis inteiras: algoritmo de **pesquisa em árvore** (aulas anteriores)
- outro método: **algoritmo dos planos de corte**
- método proposto por Ralph Gomory em 1958
- ideia: adicionar uma restrição que remova a solução atual (fracionária) da região admissível da relaxação linear, e voltar a resolver, até que a solução seja inteira
- paralelo com o modelo de DFJ: juntar restrições só quando se observa que estão a ser violadas numa solução intermédia

Nota: o algoritmo aplica-se quando no problema:

- todas as variáveis são inteiras
- todos os coeficientes das variáveis nas restrições são inteiros
- todos os termos independentes são inteiros

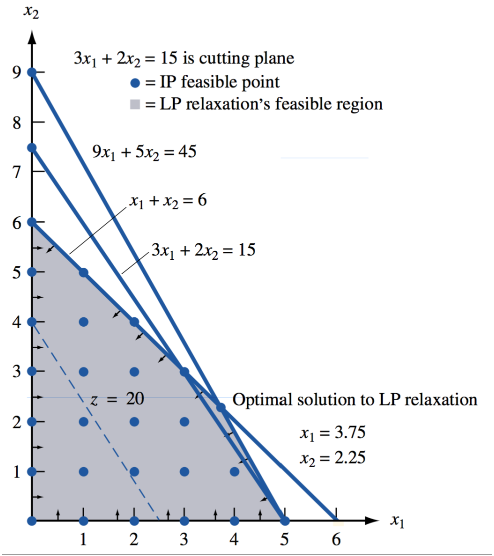
Planos de corte: exemplo

$$\begin{aligned} \text{maximizar } z = & 8x_1 + 5x_2 \\ \text{sujeito a:} \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1, x_2 \geq 0 \text{ e inteiros} \end{aligned}$$

Quadro ótimo do simplex:

z	x_1	x_2	s_1	s_2	rhs
1	0	0	1.25	0.75	41.25
0	0	1	2.25	-0.25	2.25
0	1	0	-1.25	0.25	3.75

Planos de corte: exemplo



Planos de corte: cortes válidos

Corte válido: desigualdade que se adiciona tal que:

1. a solução da relaxação linear passa a ficar fora da região admissível;
2. todas as soluções inteiras do problema original continuam na região admissível.

Planos de corte: exemplo (cont).

$$\begin{aligned} \text{maximizar } z = & 8x_1 + 5x_2 \\ \text{sujeito a: } \quad & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1, x_2 \geq 0 \text{ e inteiros} \end{aligned}$$

Quadro ótimo do simplex:

z	x_1	x_2	s_1	s_2	rhs
1	0	0	1.25	0.75	41.25
0	0	1	2.25	-0.25	2.25
0	1	0	-1.25	0.25	3.75

Planos de corte: exemplo (cont).

- escolher linha no quadro do simplex com variável básica fracionária; seja:
  $x_1 - 1.25s_1 + 0.25s2 = 3.75 \Leftrightarrow$
  $x_1 - 2s_1 + 0.75s_1 + 0s_2 + 0.25s_2 = 3 + 0.75$
- colocando à esquerda coeficientes inteiros e à direita fracionários:
  $x_1 - 2s_1 + 0s_2 - 3 = 0.75 - 0.75s_1 - 0.25s_2$
  - x_1, s_1, s_2 só podem assumir valores inteiros (porquê?)
  - lado esquerdo é inteiro
  - portanto, lado direito também, e será no máximo 0.75
  - arredondando para baixo:
- plano de corte: lado direito ≤ 0, ou seja, adiciona-se a restrição:
  $0.75 - 0.75s_1 - 0.25s_2 \leq 0$

Propriedades dos cortes de Gomory

1. qualquer solução admissível para o problema inteiro satisfaz o corte
   - seja uma solução x_1, x_2 admissível para o problema inteiro
   - x_1, x_2 são inteiras e satisfazem todas as restrições da relaxação linear
   - qualquer solução admissível deverá ter $s_1 \geq 0, s_2 \geq 0$
   - naquela linha, como $0.75 < 1$, qualquer solução admissível *inteira* deverá ter o lado direito < 1
   - ou seja, deverá satisfazer $0.75 - 0.75s_1 - 0.25s_2 \leq 0$
   - para qualquer solução inteira, $x_1 - 2s_1 + 0s_2 - 3$ é inteiro
   - portanto $0.75 - 0.75s_1 - 0.25s_2$ deverá ser um inteiro *menor que 1*
   - ou seja, o corte não remove nenhuma solução inteira admissível
2. a solução atual (no quadro do simplex) **não satisfaz** o corte
   - nesta solução, $s_1 = s_2 = 0$; não satisfaz $0.75 - 0.75s_1 - 0.25s_2 \leq 0$
   - isto resulta porque a parte fracionária $= 0.75 > 0$
   - podemos escolher *qualquer restrição* cujo lado direito no quadro ótimo do simplex seja fracionário, e "cortar" essa solução

Planos de corte: exemplo (cont.)

Quadro anterior do simplex:

z	x_1	x_2	s_1	s_2	rhs
1	0	0	1.25	0.75	41.25
0	0	1	2.25	-0.25	2.25
0	1	0	-1.25	0.25	3.75

Plano de corte a adicionar:

$$0.75 - 0.75s_1 - 0.25s_2 \leq 0 \Leftrightarrow -0.75s_1 - 0.25s_2 + s_3 = -0.75$$

Novo quadro do simplex:

z	x_1	x_2	s_1	s_2	s_3	rhs
1	0	0	1.25	0.75	0	41.25
0	0	1	2.25	-0.25	0	2.25
0	1	0	-1.25	0.25	0	3.75
0	0	0	-0.75	-0.25	1	-0.75

Algoritmo dos cortes de Gomory

- se todas as variáveis forem inteiras, a solução é ótima $\Rightarrow$ **parar**
- se houver variáveis fracionárias: **gerar corte**
  (caso haja várias: heurística: escolher a que tiver o valor mais próximo de 1/2)
- resolver com o método do simplex (dual), e recomeçar

Gomory mostrou que com este algoritmo se obtém uma solução inteira com um número finito de cortes.

Outros algoritmos de planos de corte

Exemplo: $6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10, \quad x_i \in \{0, 1\}$
Podemos reescrever a restrição numa forma mais "forte"?

- será que x_1, x_2, x_4 podem ser simultaneamente 1?
- se não, podemos remover a solução $(5/6, 1, 0, 1)$?
- a restrição $x_1 + x_2 + x_4 \leq 2$ será sempre válida?

Próxima aula

- Introdução à programação por restrições