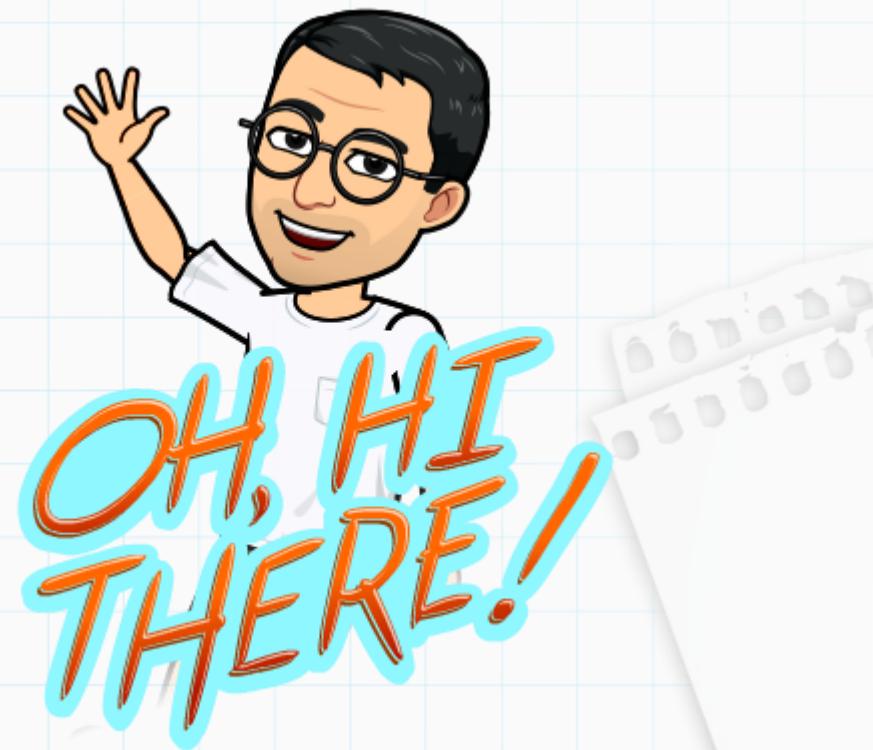


# Solvers

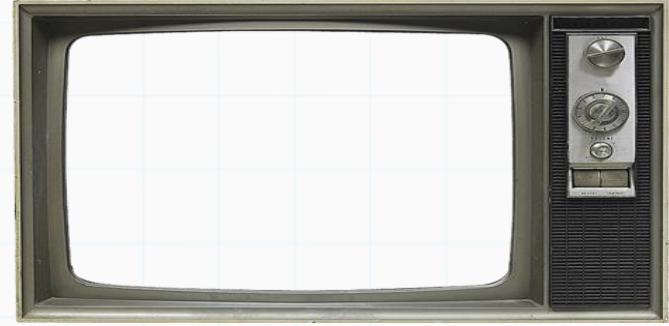
Professor : Yuri Frota

[www.ic.uff.br/~yuri/pi.html](http://www.ic.uff.br/~yuri/pi.html)

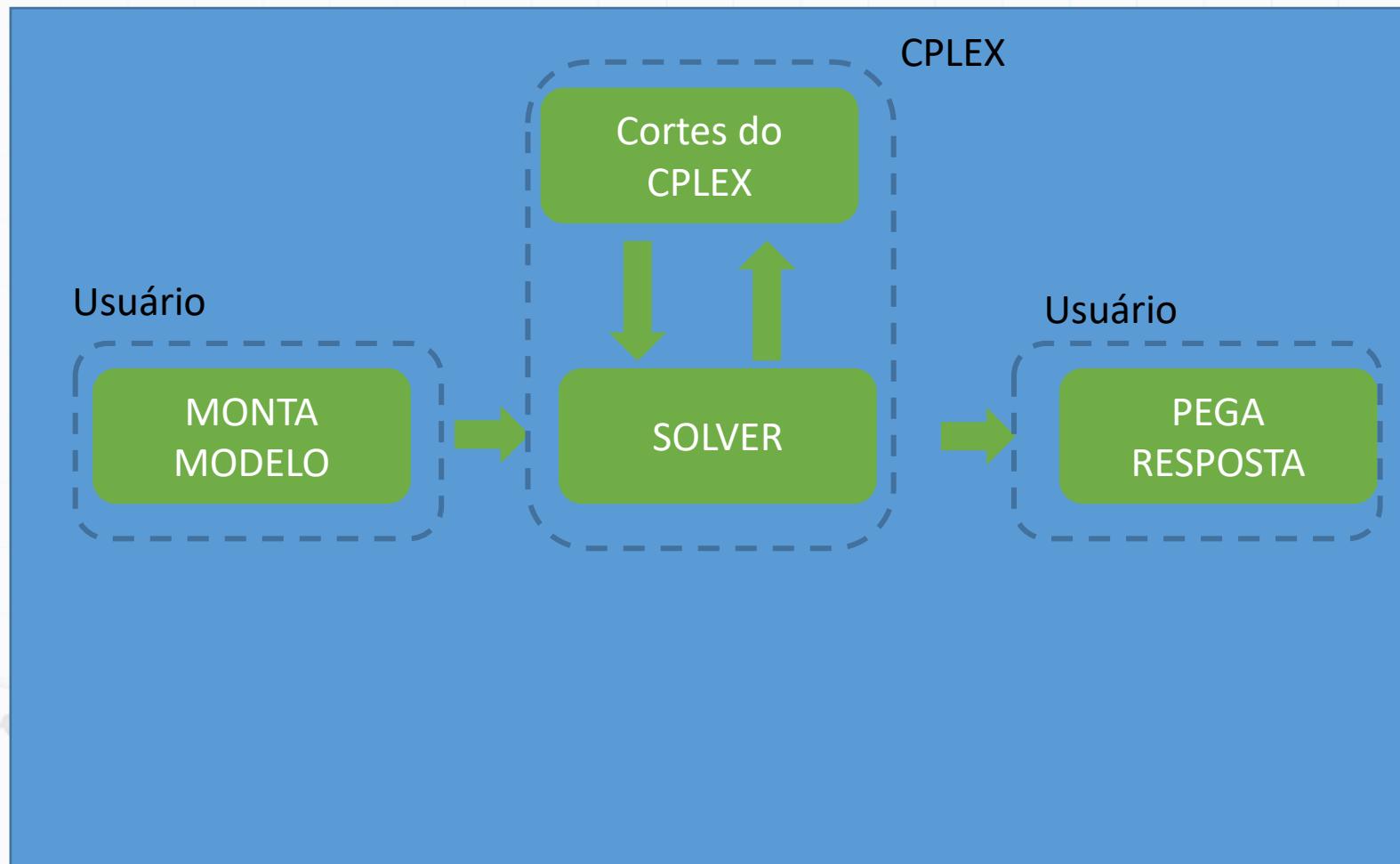
yuri@ic.uff.br



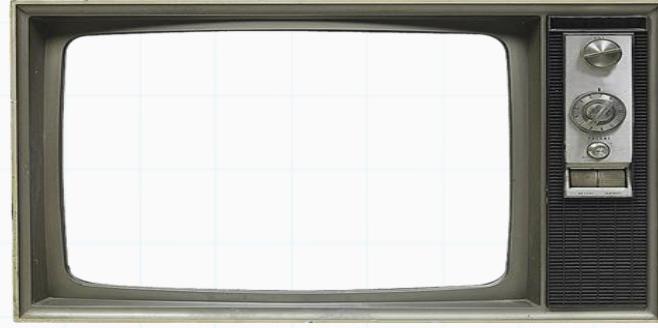
# CPLEX



Como fazer um Branch-and-cut no CPLEX de forma automática :



# CPLEX



Como fazer um Branch-and-cut no CPLEX de forma automática :

Tipos de Cortes :

Genéricos para qualquer PI (pacotes de solver tem):

- Chvatal – Gomory
- Mistos (inteiros e contínuos)
- ...

Estruturais para alguns tipos de restrição (pacotes tem alguns mais comuns):

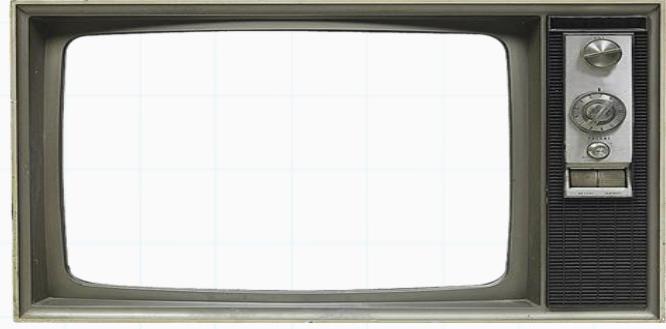
- cliques
- triangulares
- mochila
- grafos de conflito
- ...

Estruturais específicos de problemas (pacotes não tem) :

- Eliminação de subciclos
- Emparelhamentos
- Buraco ímpar
- ...

CPLEX

# CPLEX



Como fazer um Branch-and-cut no CPLEX de forma automática :

Tipos de Cortes :

Genéricos para qualquer PI (pacotes de solver tem):

- Chvatal – Gomory
- Mistos (inteiros e contínuos)
- ...

CPLEX

Estruturais para alguns tipos de restrição (pacotes tem alguns mais comuns):

- cliques
- triangulares
- mochila
- grafos de conflito
- ...

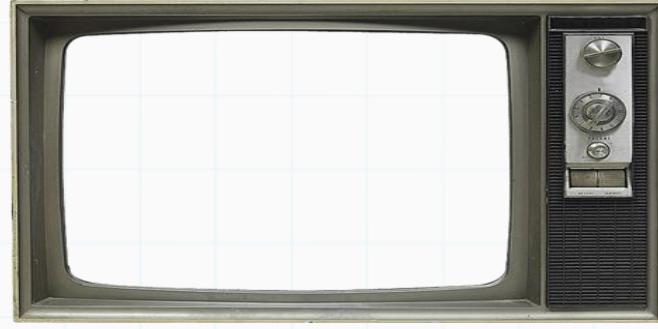


Usuário

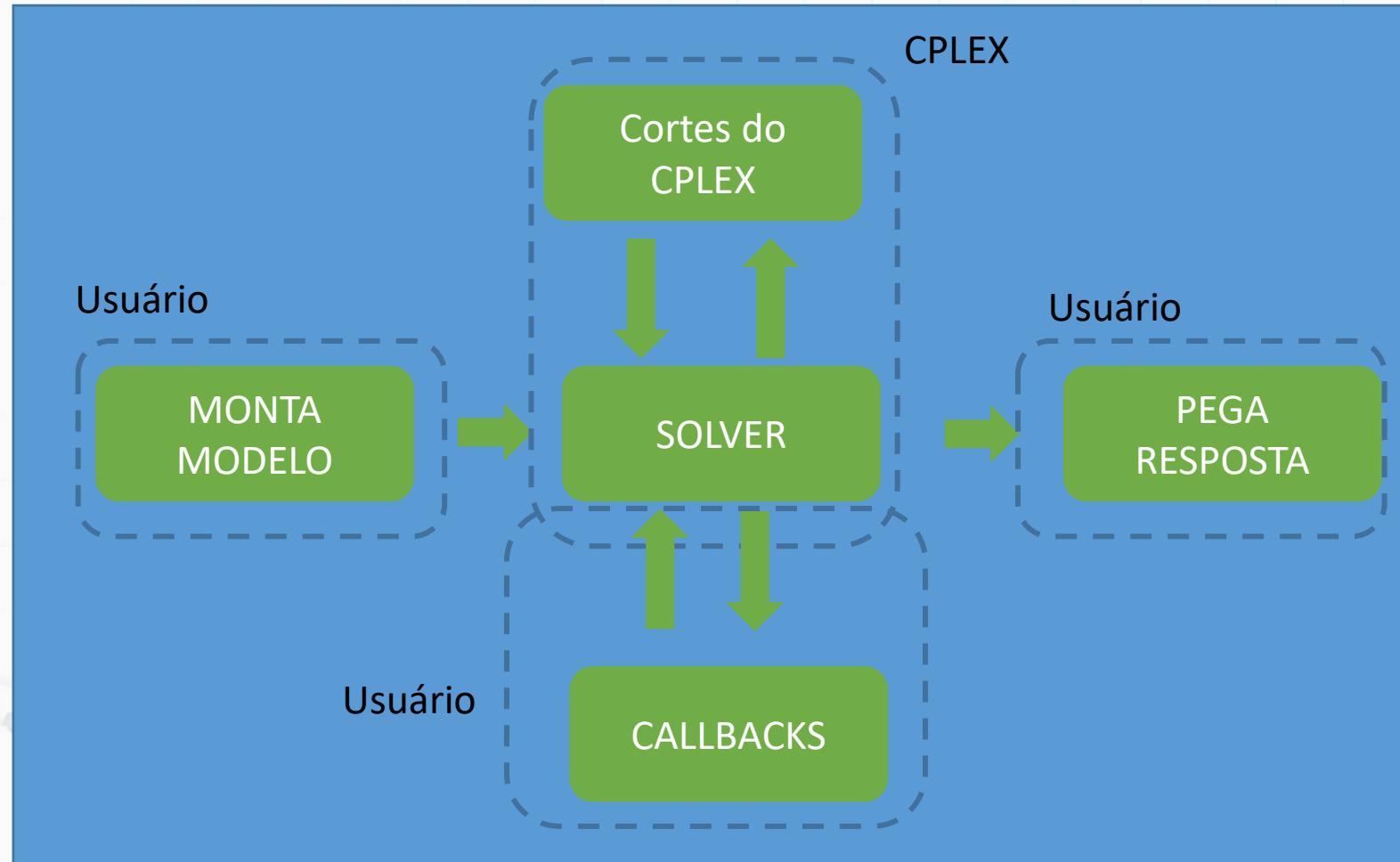
Estruturais específicos de problemas (pacotes não tem) :

- Eliminação de subciclos
- Emparelhamentos
- Buraco ímpar
- ...

# CPLEX



Como fazer um Branch-and-cut no CPLEX interagindo com o SOLVER, usando cortes específicos :

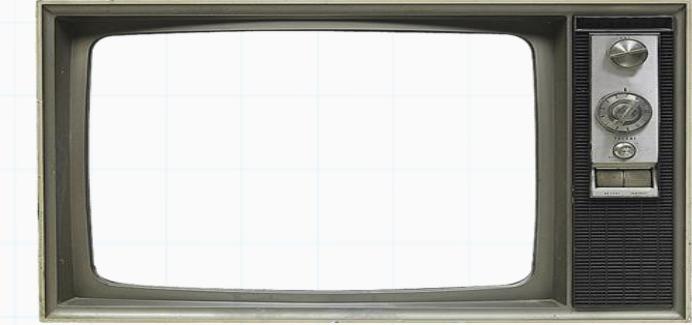


Callbacks: Funções que  
interagem com o solver  
do CPLEX

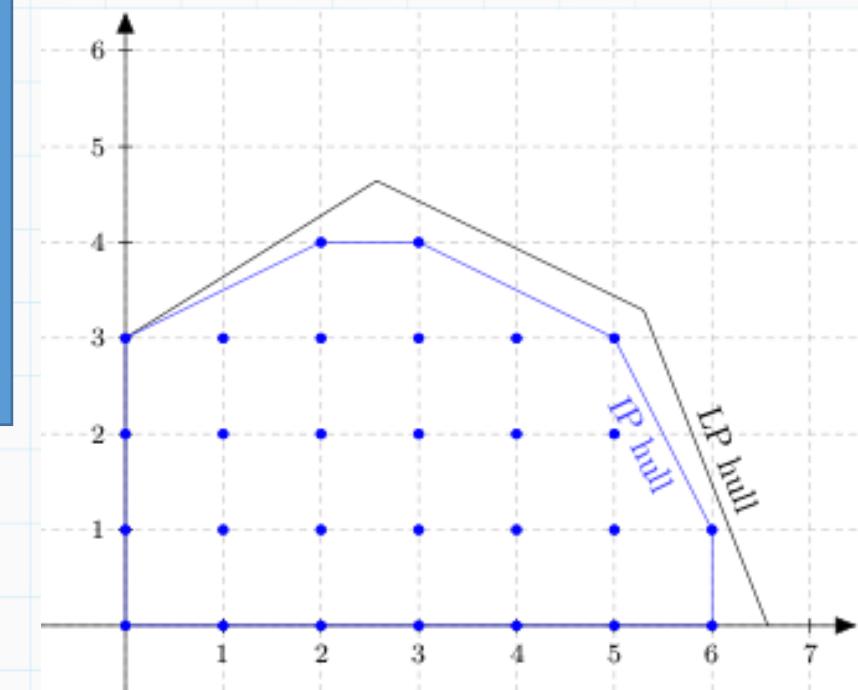
# CPLEX

Como fazer seu próprio Branch-and-Cut

Tipos de cortes:



suponha uma modelagem:

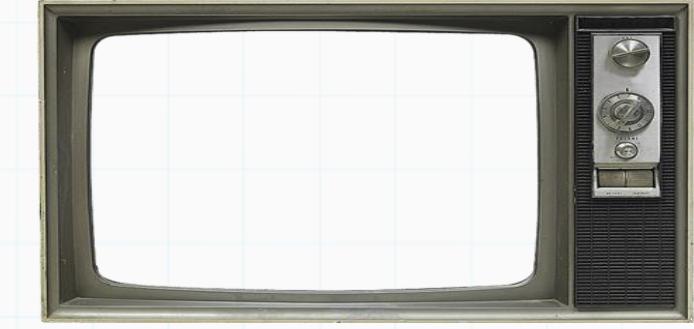


# CPLEX

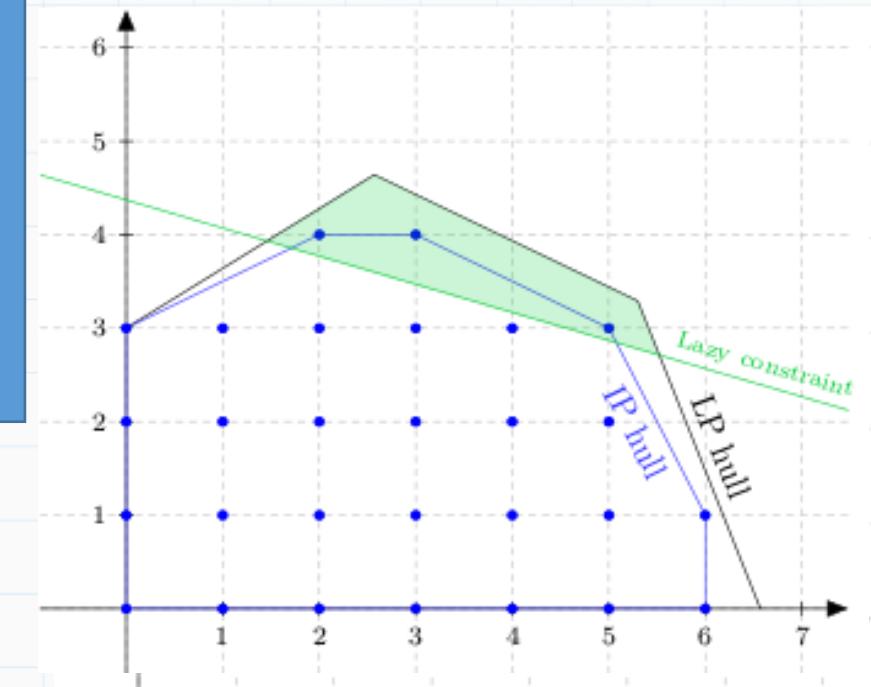
Como fazer seu próprio Branch-and-Cut

Tipos de cortes:

1. Preguiçosos (Lazy): Fazem parte do modelo original (são necessários para definir o problema), são inseridos a medida que estão violados



suponha uma modelagem:

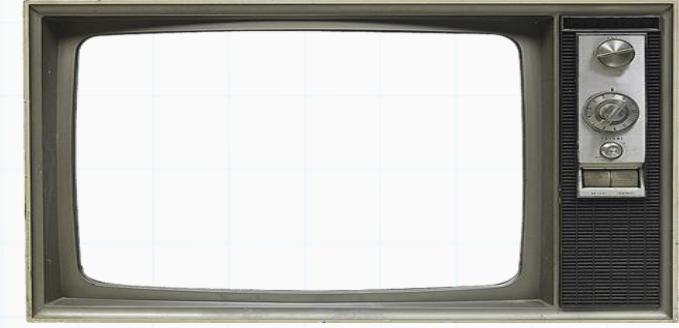


# CPLEX

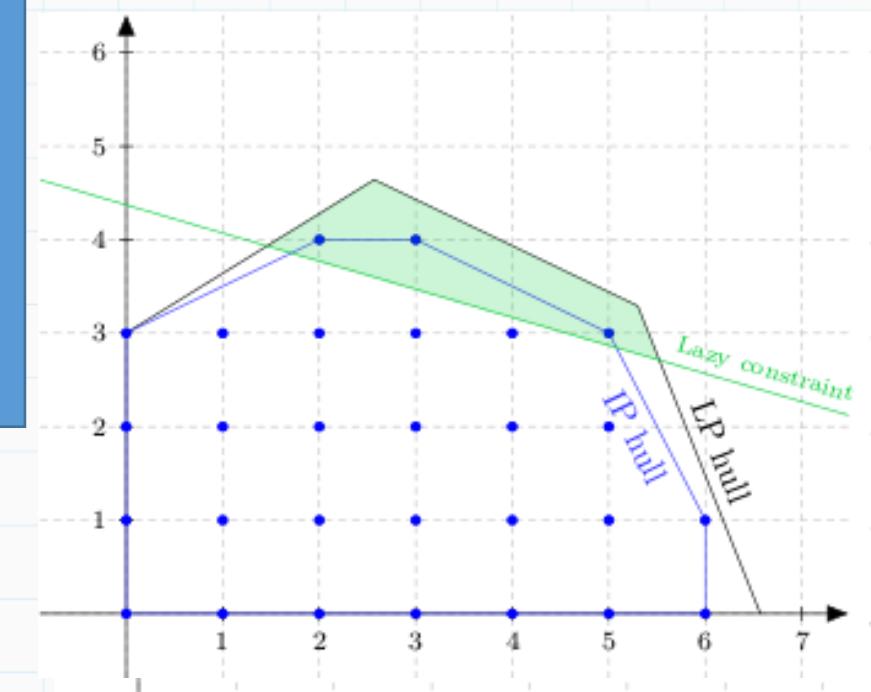
Como fazer seu próprio Branch-and-Cut

Tipos de cortes:

1. Preguiçosos (Lazy): Fazem parte do modelo original (são necessários para definir o problema), são inseridos a medida que estão violados
  - TODOS violados precisam ser gerados para termos uma solução válida inteira



suponha uma modelagem:



# CPLEX



Exemplo de cortes preguiçosos: TSP

simétrico

$$\begin{aligned} \min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\ & \sum_{(i,j) \in E} x_{ij} = 2, \quad \forall i \in V \end{aligned}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 3$$

Cortes preguiçosos a serem gerados quando violados

Modelo base

# CPLEX



Exemplo de cortes preguiçosos: TSP

simétrico

$$\min \sum_{ij \in E} c_{ij} x_{ij}$$
$$\sum_{(i,j) \in E} x_{ij} = 2, \quad \forall i \in V$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 3$$

Modelo base

Cortes preguiçosos a serem gerados quando violados

ou

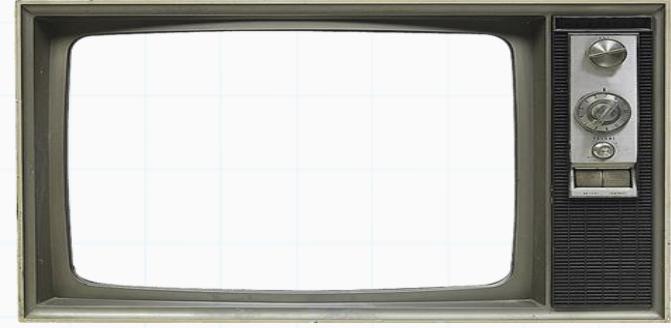
$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

# CPLEX

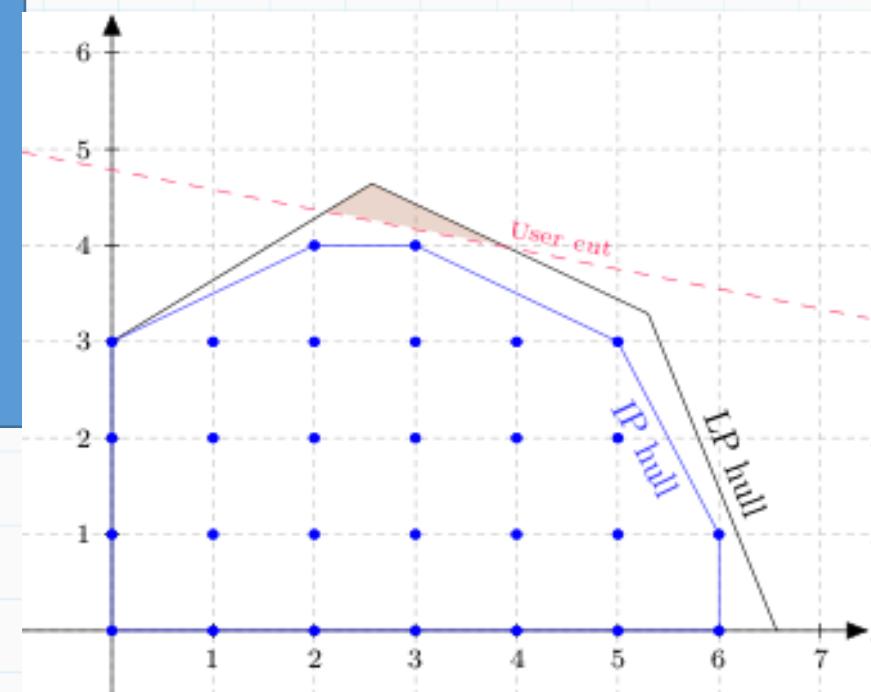
Como fazer seu próprio Branch-and-Cut

Tipos de cortes:

1. Preguiçosos (Lazy): Fazem parte do modelo original (são necessários para definir o problema), são inseridos a medida que estão violados
  - TODOS violados precisam ser gerados para termos uma solução válida inteira
2. Usuários (User): Não fazem parte do modelo original (não definem o problema), geralmente usados para fortalecer o modelo, cortar soluções fracionárias, ou evitar simetrias



suponha uma modelagem:

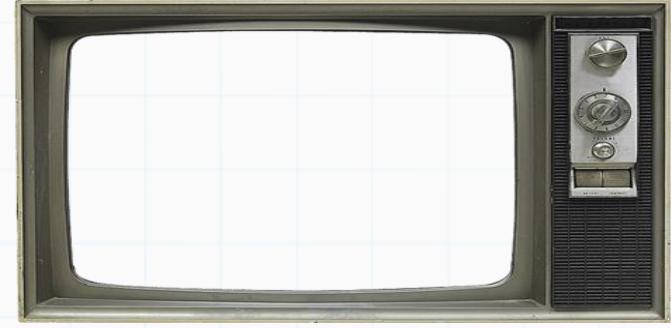


# CPLEX

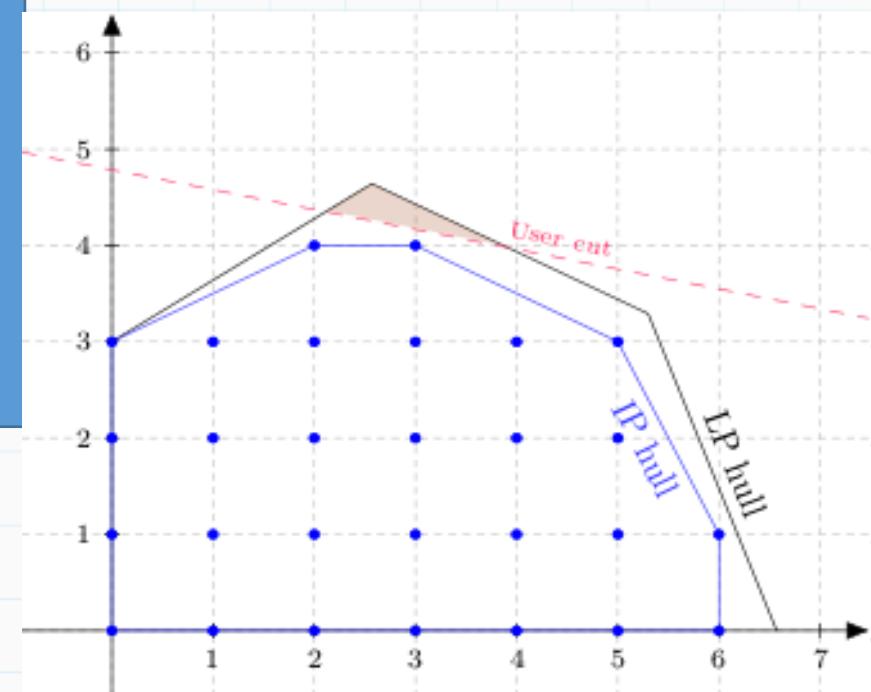
Como fazer seu próprio Branch-and-Cut

Tipos de cortes:

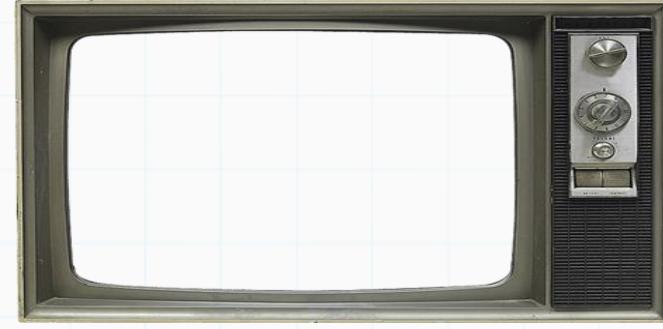
1. Preguiçosos (Lazy): Fazem parte do modelo original (são necessários para definir o problema), são inseridos a medida que estão violados
  - TODOS violados precisam ser gerados para termos uma solução válida inteira
2. Usuários (User): Não fazem parte do modelo original (não definem o problema), geralmente usados para fortalecer o modelo, cortar soluções fracionárias, ou evitar simetrias
  - Não é necessário gerar todos os cortes ( gere quantos quiser ou achar necessário)



suponha uma modelagem:



# CPLEX



Exemplo de cortes usuários: Coloração

$$\min \sum_{j \in C} w_j$$

$$\sum_{j \in C} x_{ij} = 1, \forall i \in V$$

$$x_{ij} + x_{kj} \leq 1, \forall ik \in E \text{ e } \forall j \in C$$

$$x_{ij} \leq w_j, \forall i \in V, \forall j \in C$$

$$x_{ij} \in \{0, 1\}, \forall i \in V \text{ e } \forall j \in C$$

$$w_j$$

Modelo base

$$\sum_{i \in K} x_{ij} \leq w_j, \quad \forall j \in C$$

Cortes usuários a serem gerados quando quiser, fortificam a relaxação, mas não alteram a região dos pontos inteiros

# CPLEX

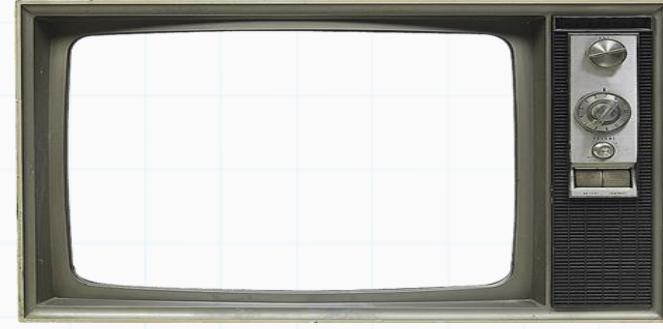
Como fazer seu próprio Branch-and-Cut

Cortes Lazy : são sempre verificados pelo CPLEX

Cortes User: Não temos essa garantia, o solver pode decidir não verificar mais



# CPLEX



Exemplo de cortes preguiçosos: TSP

simétrico

$$\begin{aligned} \min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\ & \sum_{(i,j) \in E} x_{ij} = 2, \quad \forall i \in V \end{aligned}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E$$

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 3$$

tsp2.cpp -> Cortes preguiçosos a serem gerados quando violados

tsp.cpp -> Modelo base

# CPLEX



função “use” diz que funções serão usadas como callbacks do método.

CALLBACKS: funções que serão chamadas em pontos cruciais do método. Ex: cortes, ramificações, sol. inteiras, etc

```
...
// estrutura usada para os métodos de geracao de cortes
double **residual = new double*[dim];           // grafo residual usado no corte
for(int i=0; i < dim; i++)
    residual[i] = new double[dim];
int *cutset = new int[dim];
int num_l=0,num_u=0;

// -- LAZY CUTS --
solver.use(CB_mincut(env, x, dim, residual, cutset, &num_u));

// ****
// Parametros do CPLEX
solver.setParam(IloCplex::Param::WorkMem,1024*2);           //tamanho de RAM utilizada maxima
solver.setParam(IloCplex::Param::MIP::Strategy::File, 2);   //quando a RAM acaba, 1-guarda nos na memoria e compactado 2-guarda
solver.setParam(IloCplex::Param::TimeLimit, 3600);          // tempo limite
solver.setParam(IloCplex::Param::Threads, 1);               // Numero de threads
solver.setParam(IloCplex::Param::MIP::Interval, 100);        // Log a cada N nos
solver.setOut(env.getNullStream());                         // Desabilitando saida do cplex

// Parametros do B&C
solver.setParam(IloCplex::Param::Preprocessing::Presolve, 0); // desliga o preprocessamento, 1-ligado 0-desligado
solver.setParam(IloCplex::Param::MIP::Limits::CutsFactor, 0); // número de cortes que o CPLEX pode gerar, 0-desliga os cortes
solver.setParam(IloCplex::Param::MIP::Strategy::HeuristicFreq, -1); // heurísticas primas do CPLEX, -1-desliga 0-Liga

...
|
```

# CPLEX



função “use” diz que funções serão usadas como callbacks do método.

CALLBACKS: funções que serão chamadas em pontos cruciais do método. Ex: cortes, ramificações, sol. inteiras, etc

```
...
// estrutura usada para os métodos de geracao de cortes
double **residual = new double*[dim]; // grafo residual usado no corte
for(int i=0; i < dim; i++)
    residual[i] = new double[dim];
int *cutset = new int[dim];
int num_l=0,num_u=0;

// -- LAZY CUTS --
solver.use(CB_mincut(env, x, dim, residual, cutset, &num_u));

// ****
// Parametros do CPLEX
solver.setParam(IloCplex::Param::WorkMem,1024*2); //tamanho de RAM utilizada maxima
solver.setParam(IloCplex::Param::MIP::Strategy::File, 2); //quando a RAM acaba, 1-guarda nos na memoria e compactado 2-guarda
solver.setParam(IloCplex::Param::TimeLimit, 3600); // tempo limite
solver.setParam(IloCplex::Param::Threads, 1); // Numero de threads
solver.setParam(IloCplex::Param::MIP::Interval, 100); // Log a cada N nos
solver.setOut(env.getNullStream()); // Desabilitando saida do cplex

// Parametros do B&C
solver.setParam(IloCplex::Param::Preprocessing::Presolve, 0); // desliga o preprocessamento, 1-ligado 0-desligado
solver.setParam(IloCplex::Param::MIP::Limits::CutsFactor, 0); // número de cortes que o CPLEX pode gerar, 0-desliga os cortes
solver.setParam(IloCplex::Param::MIP::Strategy::HeuristicFreq, -1); // heurísticas primas do CPLEX, -1-desliga 0-Liga
...
|
```

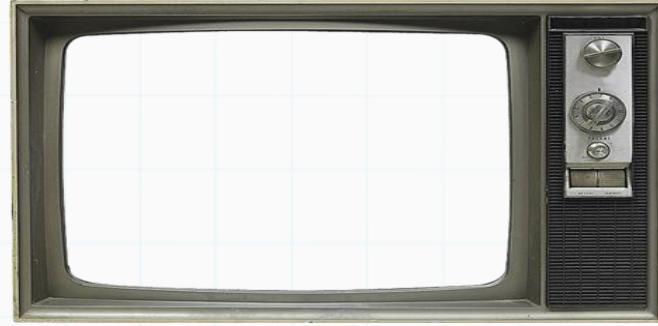
# CPLEX



Desligamos a geração de corte automática para ver melhor o impacto dos cortes gerados

```
...  
  
// estrutura usada para os métodos de geracao de cortes  
double **residual = new double*[dim]; // grafo residual usado no corte  
for(int i=0; i < dim; i++)  
    residual[i] = new double[dim];  
int *cutset = new int[dim];  
int num_l=0,num_u=0;  
  
// -- LAZY CUTS --  
solver.use(CB_mincut(env, x, dim, residual, cutset, &num_u));  
  
// *****  
  
// Parametros do CPLEX  
solver.setParam(IloCplex::Param::WorkMem, 1024*2); //tamanho de RAM utilizada maxima  
solver.setParam(IloCplex::Param::MIP::Strategy::File, 2); //quando a RAM acaba, 1-guarda nos na memoria e compactado 2-guarda  
solver.setParam(IloCplex::Param::TimeLimit, 3600); // tempo limite  
solver.setParam(IloCplex::Param::Threads, 1); // Numero de threads  
solver.setParam(IloCplex::Param::MIP::Interval, 100); // Log a cada N nos  
solver.setOut(env.getNullStream()); // Desabilitando saida do cplex  
  
// Parametros do B&C  
solver.setParam(IloCplex::Param::Preprocessing::Presolve, 0); // desliga o preprocessamento, 1-ligado 0-desligado  
solver.setParam(IloCplex::Param::MIP::Limits::CutsFactor, 0); // número de cortes que o CPLEX pode gerar, 0-desliga os cortes  
solver.setParam(IloCplex::Param::MIP::Strategy::HeuristicFreq, -1); // heurísticas primas do CPLEX, -1-desliga 0-Liga  
  
...|
```

# CPLEX



MACROS que definem as funções de corte (lazy e user) como “callbacks de cortes”

- definidas antes do programa principal

```
//ILOUSERCUTCALLBACK0(name)
//ILOUSERCUTCALLBACK1(name, type1, x1)
//ILOUSERCUTCALLBACK2(name, type1, x1, type2, x2)
//ILOUSERCUTCALLBACK3(name, type1, x1, type2, x2, type3, x3)

//IOLAZYCONSTRAINTCALLBACK0(name)
//IOLAZYCONSTRAINTCALLBACK1(name, type1, x1)
//IOLAZYCONSTRAINTCALLBACK2(name, type1, x1, type2, x2)
//IOLAZYCONSTRAINTCALLBACK3(name, type1, x1, type2, x2, type3, x3)

IOLAZYCONSTRAINTCALLBACK5(CB_mincut, IloArray<IloBoolVarArray>, x, int, dim, double**, residual, int*, cutset, int*, numc)
//ILOUSERCUTCALLBACK5(CB_mincut, IloArray<IloBoolVarArray>, x, int, dim, double**, residual, int*, cutset, int*, numc)
{
    // x          -> vetor das variaveis do problema
    // dim         -> dimensao do problema (numero de cidades)
    // residual   -> armazena grafo com a solucao corrente
    // cutset     -> conjunto S a ser encontrado
    // numc       -> contador de numero de cortes gerados
```

vai até 7 parâmetros

As macros recebem o nome da função e seus parâmetros

```
// -- LAZY CUTS --
solver.use(CB_mincut(env, x, dim, residual, cutset, &num_u));
```

# CPLEX

```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

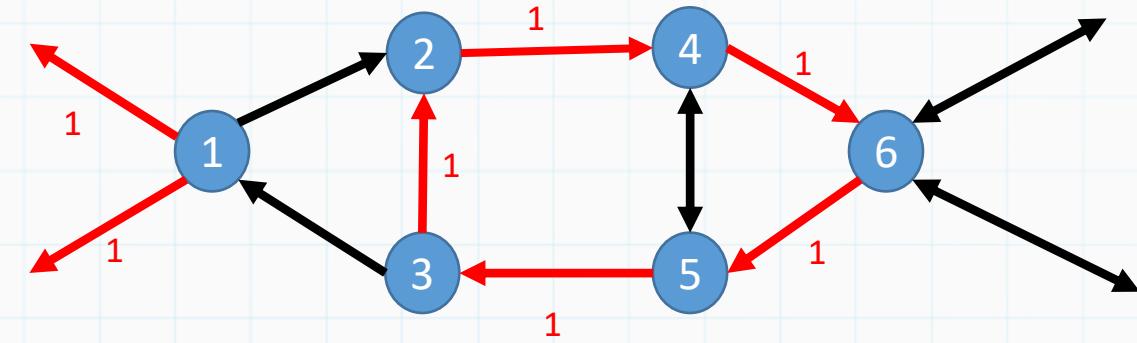
for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

O CPLEX chama a função de corte depois de resolver a relaxação linear, e antes da ramificação !

a função monta o grafo residual. Ex:



e encontra um conjunto  $S$  onde a restrição de corte está violada

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$

# CPLEX

```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

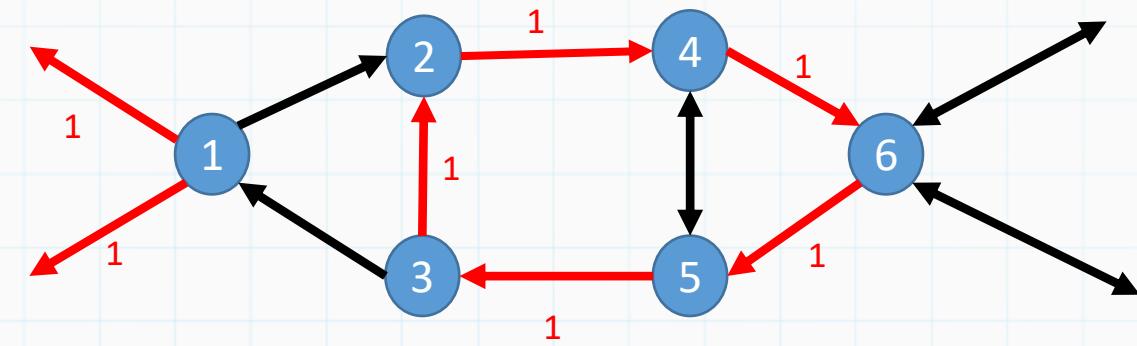
for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

Corte = Busca em largura(Grafo residual)

a função monta o grafo residual. Ex:



e encontra um conjunto  $S$  onde a restrição de corte está violada

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$

Veja que dado uma solução inteira, é fácil encontrar subciclos, basta fazer uma busca em largura no grafo residual identificando os ciclos que não pegam todos os vértices

# CPLEX

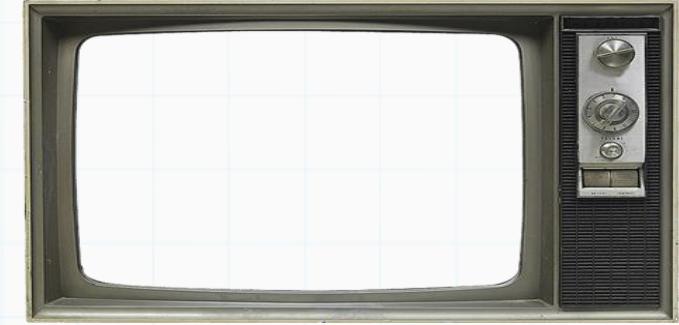
```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

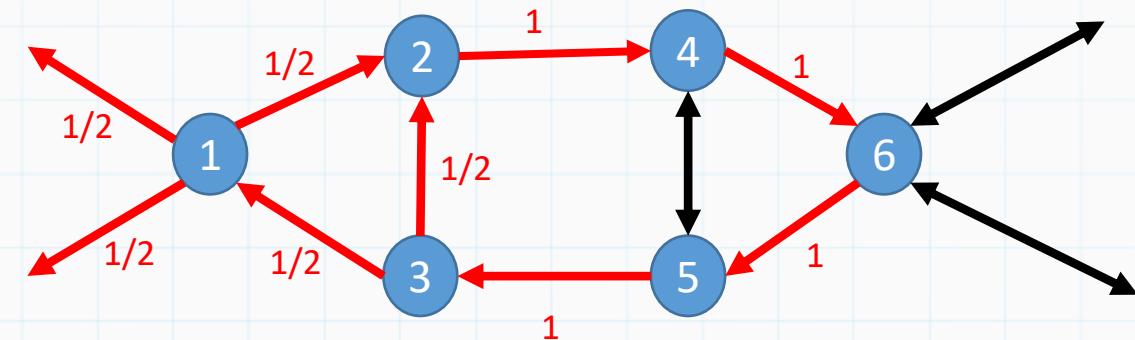
        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

O corte de subciclo pode ser  
usando como um corte  
fracionário também !!!



a função monta o grafo residual. Ex:



e encontra um conjunto  $S$  onde a restrição de corte está violada

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$

# CPLEX

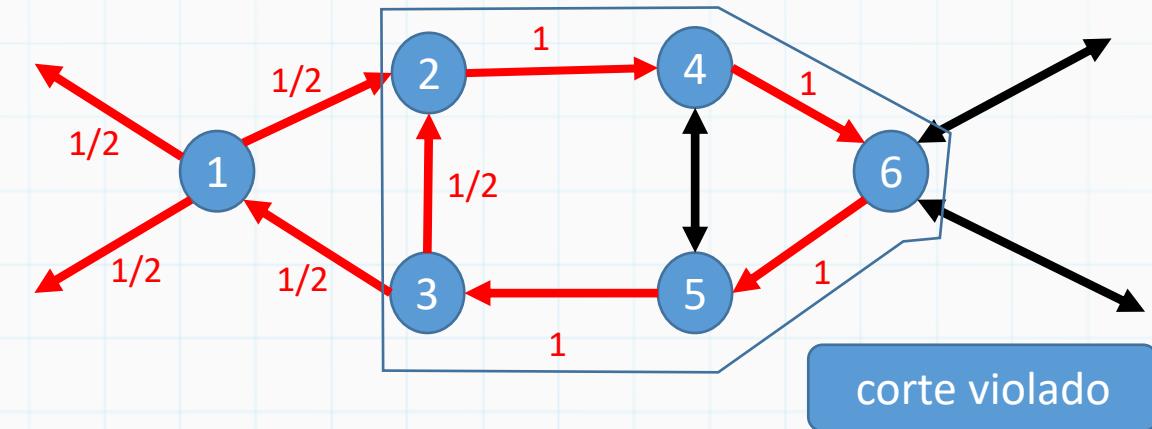
```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

a função monta o grafo residual. Ex:



e encontra um conjunto  $S$  onde a restrição de corte está violada

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$

# CPLEX

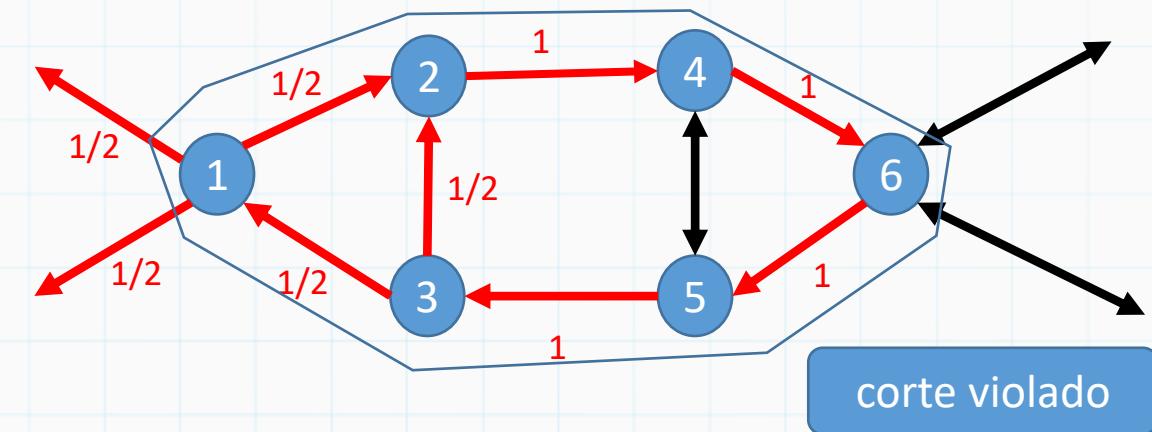
```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

a função monta o grafo residual. Ex:



e encontra um conjunto  $S$  onde a restrição de corte está violada

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$

# CPLEX

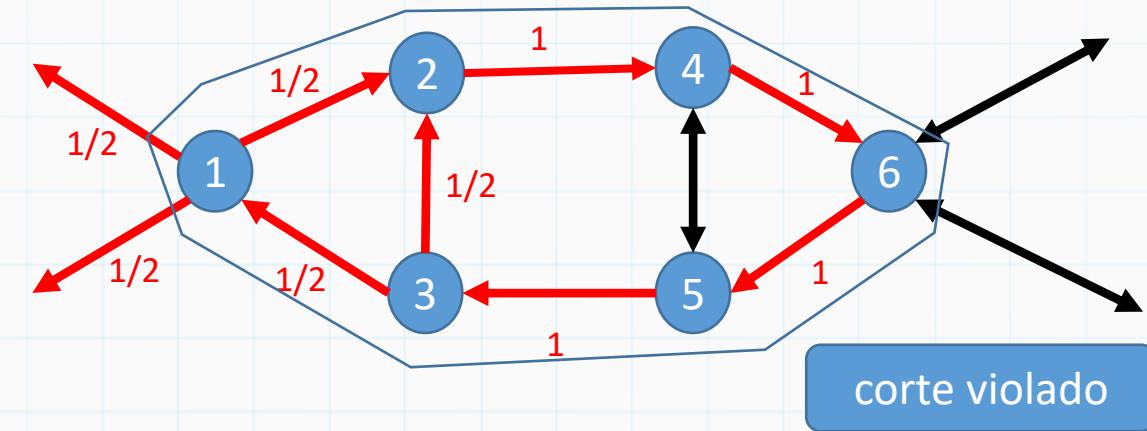
```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

a função monta o grafo residual. Ex:



e encontra um conjunto  $S$  onde a restrição de corte está violada

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$



Mas como encontrar esses cortes violados ???

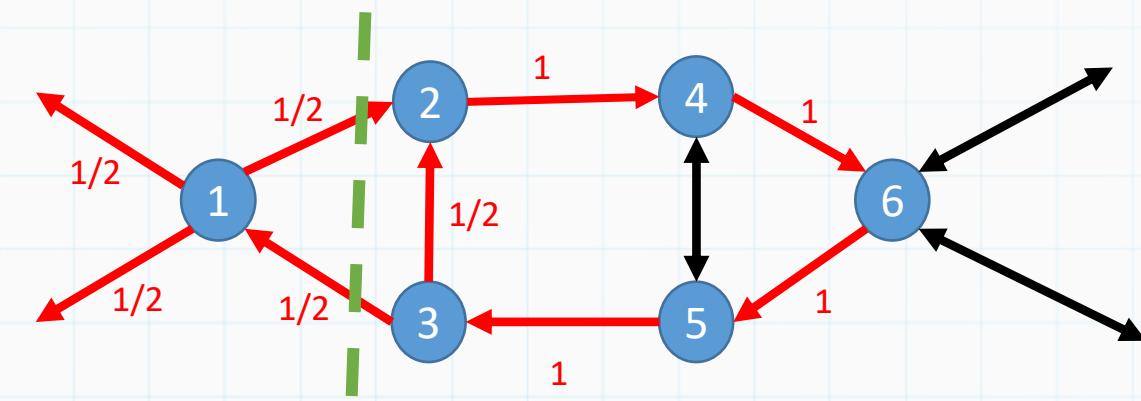
# CPLEX

```
// cria grafo da solucao corrente fracionaria
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte sera armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```



Veja que um corte no grafo residual com valor menor que 2, então existe um corte violado do tipo:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

# CPLEX

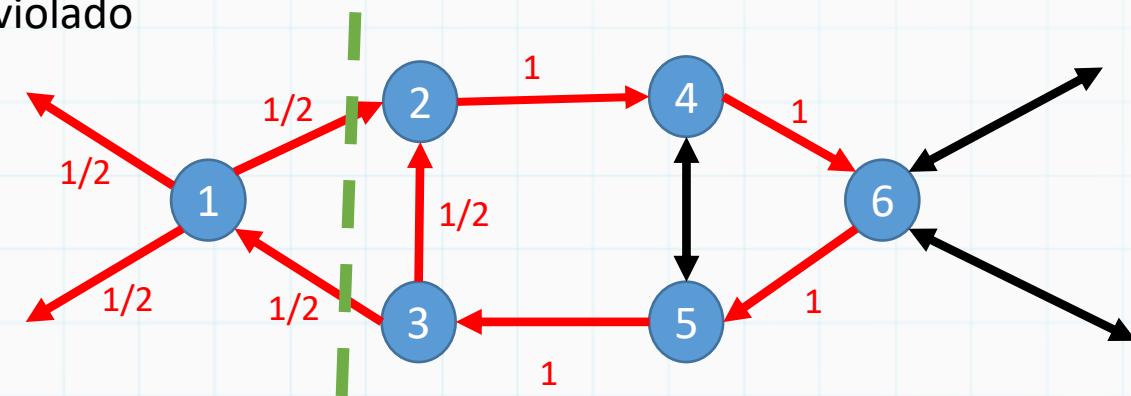
```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

A heurística max\_back\_heuristic é um método de encontrar o corte mínimo no grafo residual, pois veja que se temos um corte mínimo com valor menor que 2, então existe um corte violado



$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

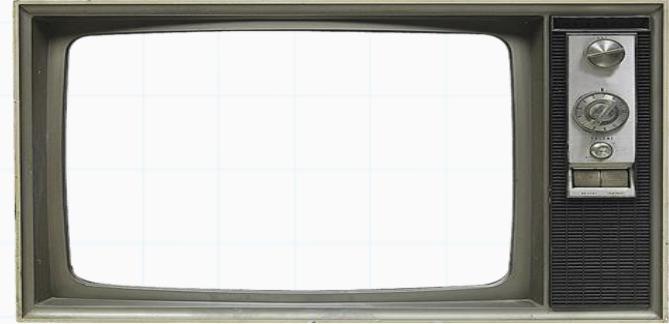
# CPLEX

```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

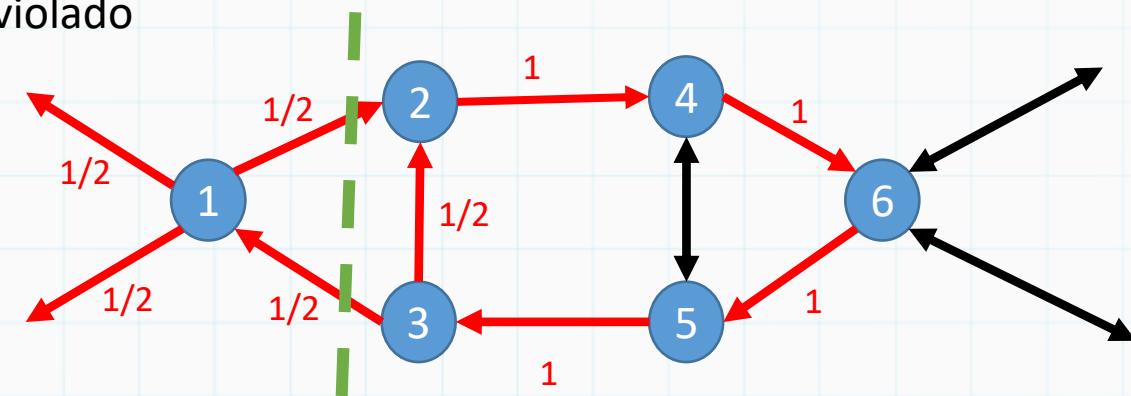
for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```



A heurística max\_back\_heuristic é um método de encontrar o corte mínimo no grafo residual, pois veja que se temos um corte mínimo com valor menor que 2, então existe um corte violado



$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

Caso o corte mínimo seja  $\geq 2$  sabemos então que não existe nenhum corte violado.

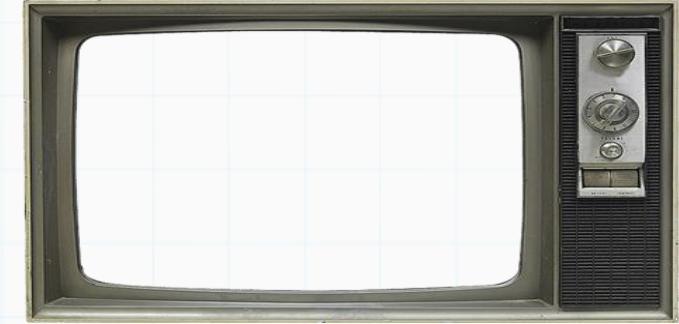
# CPLEX

```
// cria grafo da solução corrente fracionária
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

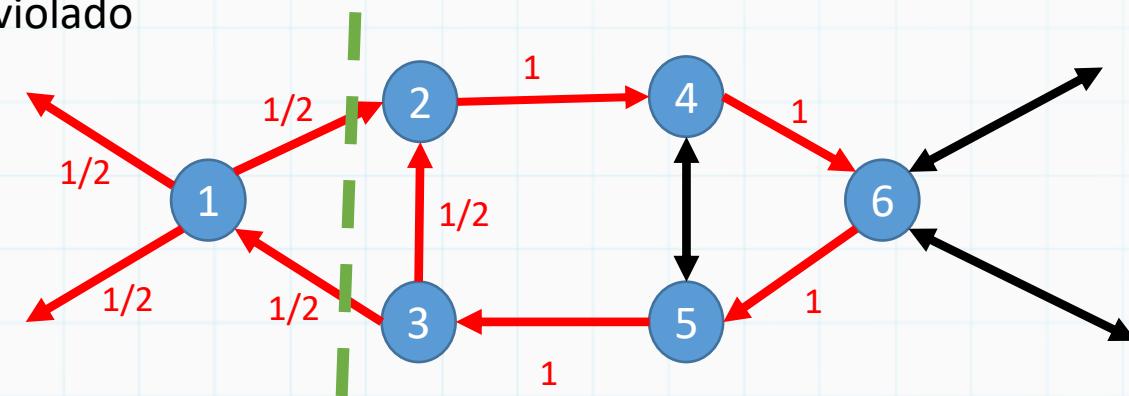
for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte será armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```



A heurística max\_back\_heuristic é um método de encontrar o corte mínimo no grafo residual, pois veja que se temos um corte mínimo com valor menor que 2, então existe um corte violado



$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

Caso o corte mínimo seja  $\geq 2$  sabemos então que não existe nenhum corte violado. Logo o problema de separação pode ser resolvido polinomialmente por um algoritmo de fluxo máximo de  $O(n^2m)$ .

Min. Cut.  $\Leftrightarrow$  Max Flow

# CPLEX

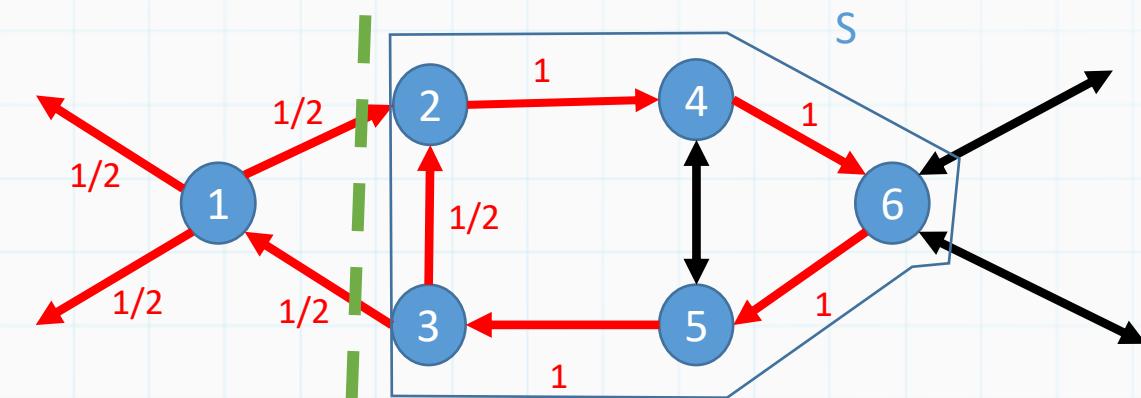
```
// cria grafo da solucao corrente fracionaria
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);
        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte sera armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;
```

Veja que todo corte de subciclo  $\geq 2$ , equivale a um corte  $\leq |S|-1$  de um dos lados do corte



$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$

Se corte  $< 2$  então o conjunto  $S$  vai ter que ter um número de arcos maior que  $|S|-1$  para conseguir satisfazer as restrições de grau

# CPLEX

```
// cria grafo da solucao corrente fracionaria
for(int i=0; i < dim; i++) {
    for(int j=i; j < dim; j++) {
        residual[i][j] = 0;
        residual[j][i] = 0;
    }
}

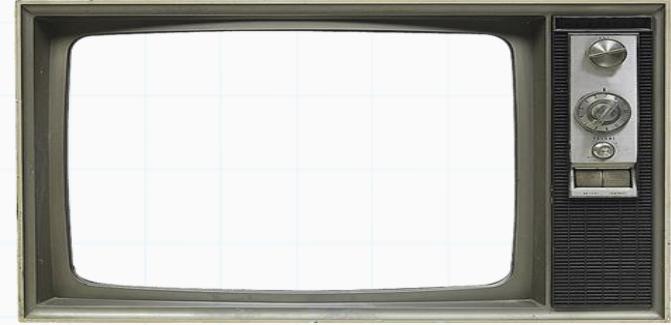
for(int i=0; i < dim-1; i++) {
    for(int j=i+1; j < dim; j++) {
        float x_ij = getValue(x[i][j]);

        if (x_ij < PRECISAO)
            x_ij = 0;

        residual[i][j] = x_ij;
        residual[j][i] = x_ij;
    }
}

// onde o corte sera armazenado (conjunto S)
for(int i=0; i < dim; i++)
    cutset[i] = 0;

// metodo para corte minimo
double cutvalue;
cutvalue = max_back_heuristic(dim, residual, cutset);
```



Heurística de Max Flow

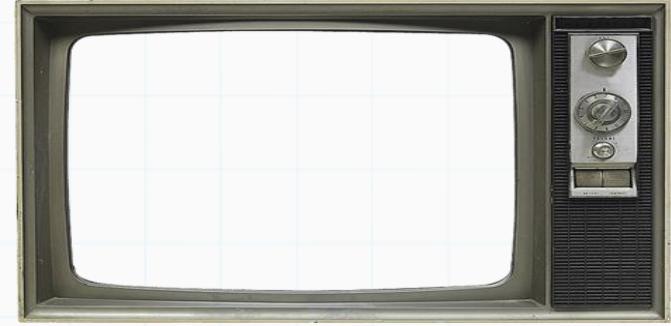
# CPLEX

```
// metodo para corte minimo
double cutvalue;
cutvalue = max_back_heuristic(dim, residual, cutset);

// gera corte
if(cutvalue + PRECISAO < 2) {
    double cutset_size = 0;
    for(int i=0; i < dim; i++){
        if (cutset[i]==1){
            cutset_size += cutset[i];
        }
    }

    IloExpr corte(env);
    double val=0;
    for(int i=0; i < dim; i++) {
        for(int j=i+1; j < dim; j++) {
            if(cutset[i] == 1 && cutset[j] == 1) {
                corte += x[i][j];
                val += getValue(x[i][j]);
            }
        }
    }
    add(corte <= cutset_size-1).end();
    *numc = *numc + 1;
}
```

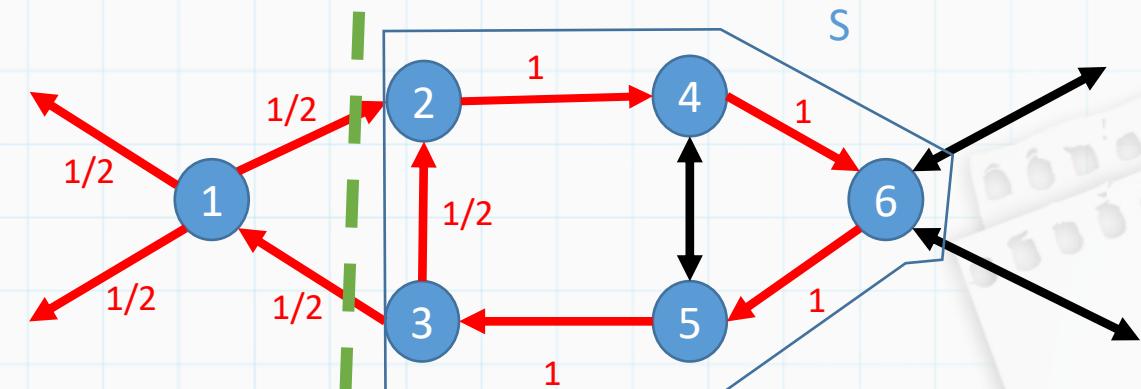
Então, dado que  
identificamos o corte...



Soma dos arcos entrando e saindo de S

Corte gerado:

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$



# CPLEX

```
// metodo para corte minimo
double cutvalue;
cutvalue = max_back_heuristic(dim, residual, cutset);

// gera corte
if(cutvalue + PRECISAO < 2) {

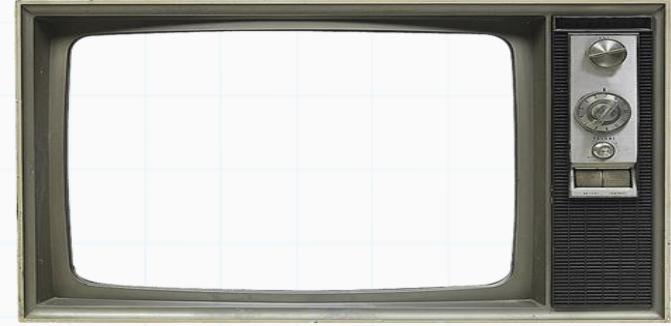
    double cutset_size = 0;
    for(int i=0; i < dim; i++){
        if (cutset[i]==1){
            cutset_size += cutset[i];
        }
    }

    IloExpr corte(env);
    double val=0;
    for(int i=0; i < dim; i++) {
        for(int j=i+1; j < dim; j++) {
            if(cutset[i] == 1 && cutset[j] == 1) {
                corte += x[i][j];
                val += getValue(x[i][j]);
            }
        }
    }

    add(corte <= cutset_size-1).end();
    *numc = *numc + 1;
}
```

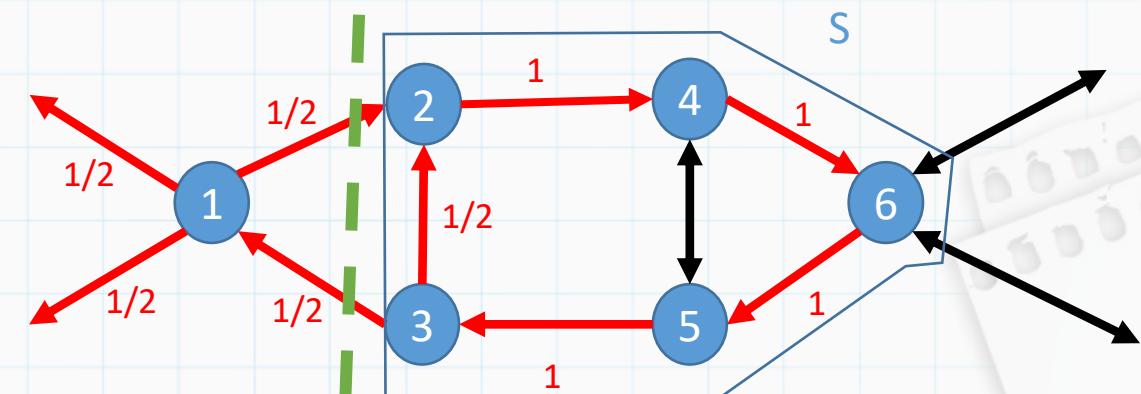
determina  $|S|$

Então, dado que  
identificamos o corte...



Corte gerado:

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$



# CPLEX

```
// metodo para corte minimo
double cutvalue;
cutvalue = max_back_heuristic(dim, residual, cutset);

// gera corte
if(cutvalue + PRECISAO < 2) {

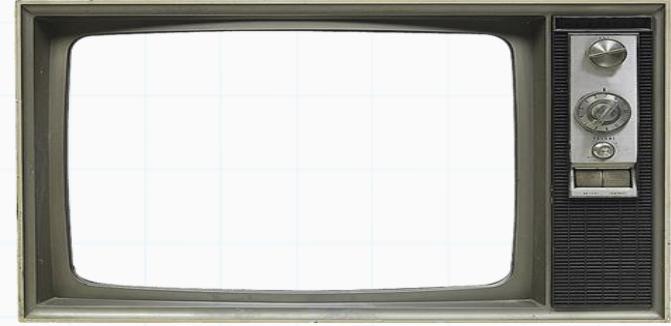
    double cutset_size = 0;
    for(int i=0; i < dim; i++){
        if (cutset[i]==1){
            cutset_size += cutset[i];
        }
    }

    IloExpr corte(env);
    double val=0;
    for(int i=0; i < dim; i++) {
        for(int j=i+1; j < dim; j++) {
            if(cutset[i] == 1 && cutset[j] == 1) {
                corte += x[i][j];
                val += getValue(x[i][j]);
            }
        }
    }

    add(corte <= cutset_size-1).end();
    *numc = *numc + 1;
}
```

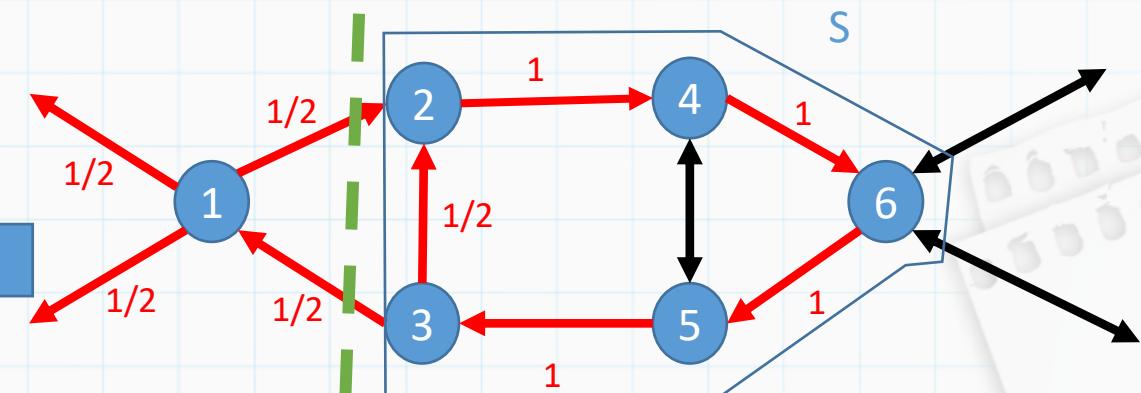
monta corte

Então, dado que  
identificamos o corte...



Corte gerado:

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$



# CPLEX

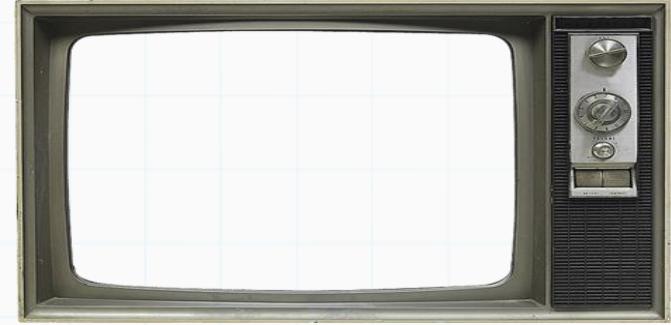
```
// metodo para corte minimo
double cutvalue;
cutvalue = max_back_heuristic(dim, residual, cutset);

// gera corte
if(cutvalue + PRECISAO < 2) {

    double cutset_size = 0;
    for(int i=0; i < dim; i++){
        if (cutset[i]==1){
            cutset_size += cutset[i];
        }
    }

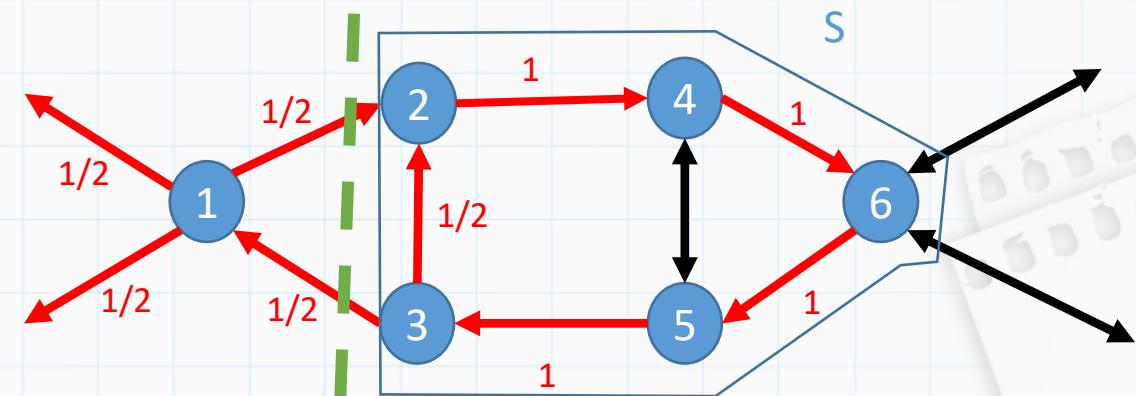
    IloExpr corte(env);
    double val=0;
    for(int i=0; i < dim; i++) {
        for(int j=i+1; j < dim; j++) {
            if(cutset[i] == 1 && cutset[j] == 1) {
                corte += x[i][j];
                val += getValue(x[i][j]);
            }
        }
    }
    add(corte <= cutset_size-1).end();
    *numc = *numc + 1;
}
```

Então, dado que  
identificamos o corte...



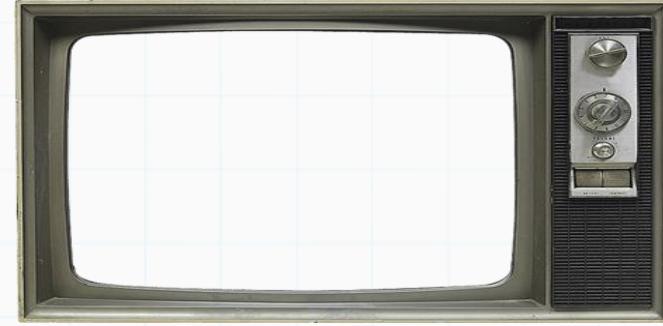
Corte gerado:

$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad \forall S \subset V, |S| \geq 2$$



não precisa colocar “model.add”, apenas “add”  
porque estamos dentro do callback

# CPLEX



Podemos acessar informações dentro do “callback” também

```
add(corte <= cutset_size-1).end();
*numc = *numc + 1;

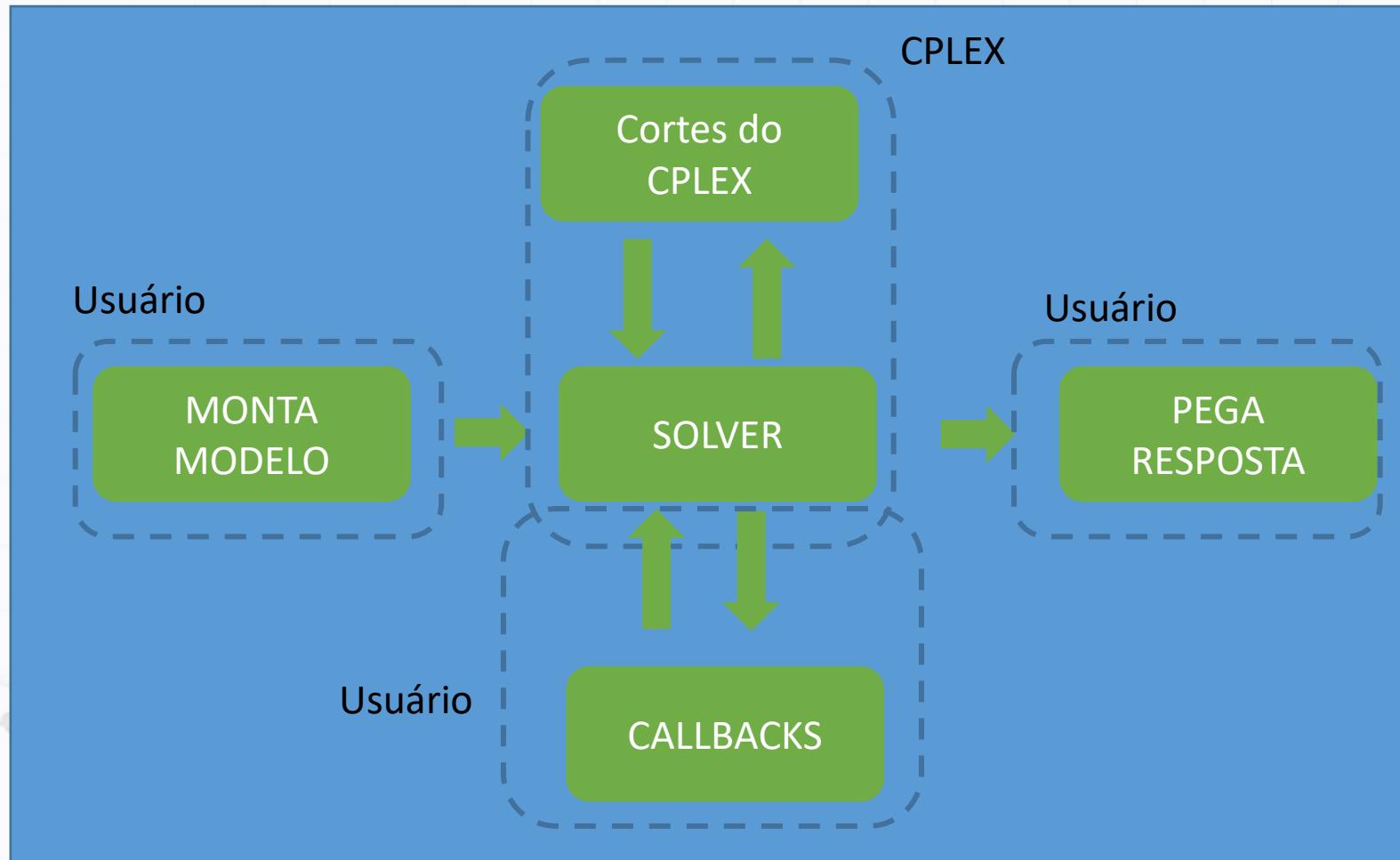
// Libera memoria
corte.end();

double ub      = getIncumbentObjValue();    // retorna a melhor solucao inteira (limite primal)
double lb      = getBestObjValue();          // retorna o melhor limite dual
double rlx     = getObjValue();              // quando chamada dentro do callback, retorna o valor da relaxacao do noh
double nNodes = getNremainingNodes();        // retorna o numero restante de nos a serem analisados
cout<<"--- LAZY CUT:<<"relax=<<rlx<<"\t bounds=<<lb<<"<->"<<ub<<"\t n_rest=<<nNodes<<"\t Ucuts=<<*numc<<endl;
```

# CPLEX



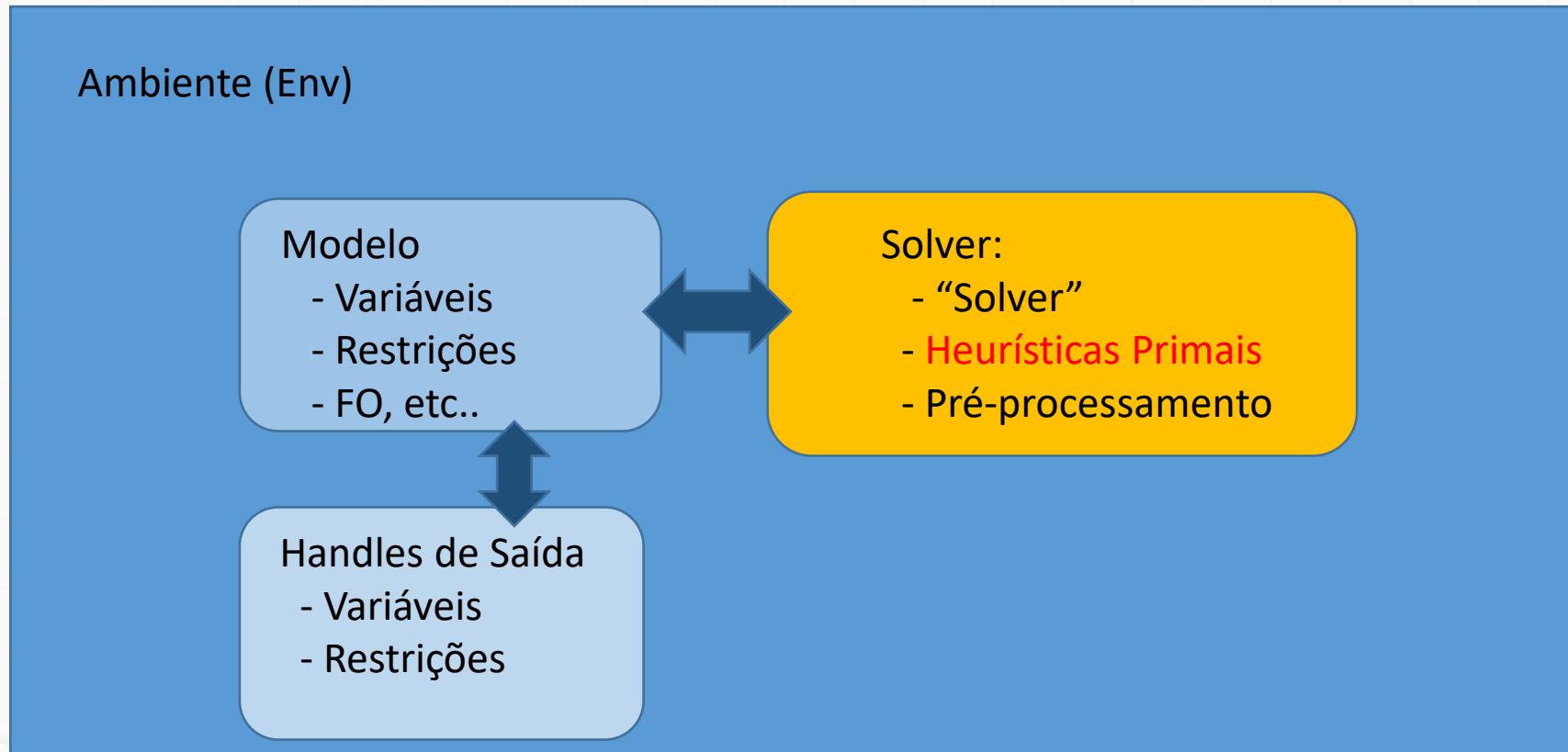
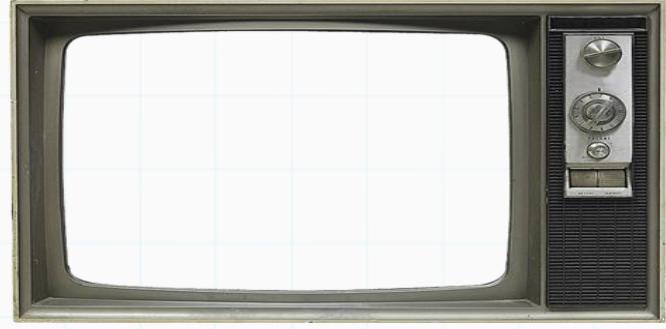
Outros callbacks ?



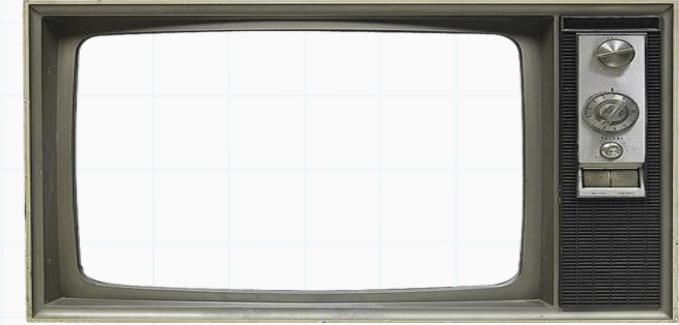
- Solução inteira
- Escolha de ramificação
- Heurística
- Pré-processamento
- Atualização da melhor solução
- Seleção do nó a ser explorado
- Muuuuuuito mais...

# CPLEX

E como as Heurísticas primais automáticas do CPLEX se comportam ?

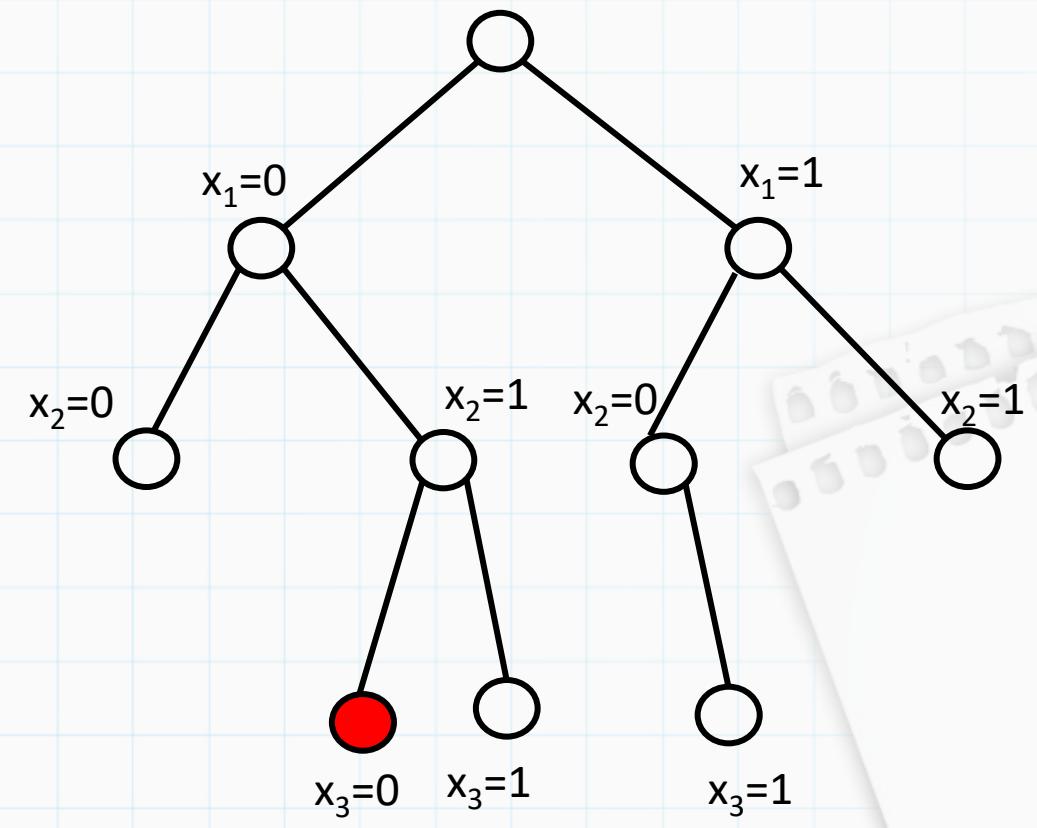


# CPLEX

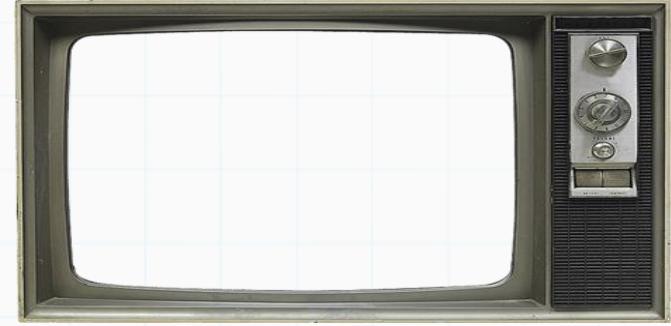


E como as Heurísticas primais automáticas do CPLEX se comportam ?

- Chamada pelo CPLEX sempre que uma solução ótima de um subproblema é encontrada

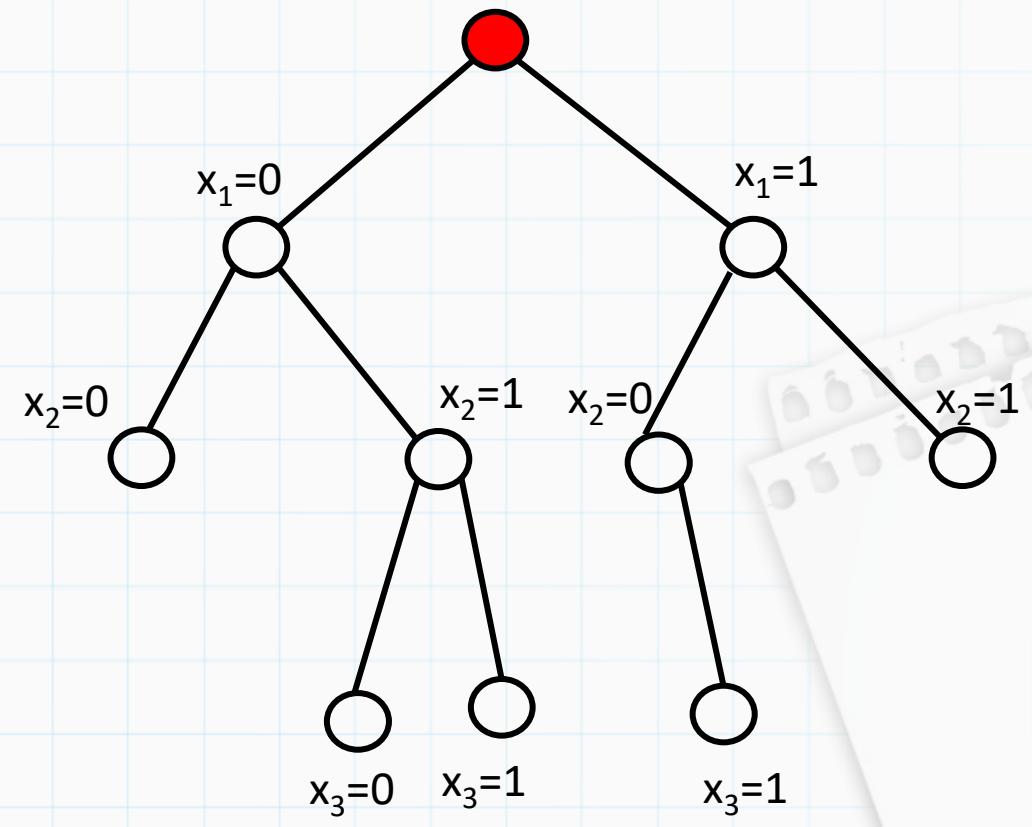


# CPLEX



E como as Heurísticas primais automáticas do CPLEX se comportam ?

- Chamada pelo CPLEX sempre que uma solução ótima de um subproblema é encontrada
- No nó raiz, é chamada após cada iteração da rotina dos cortes automáticos.

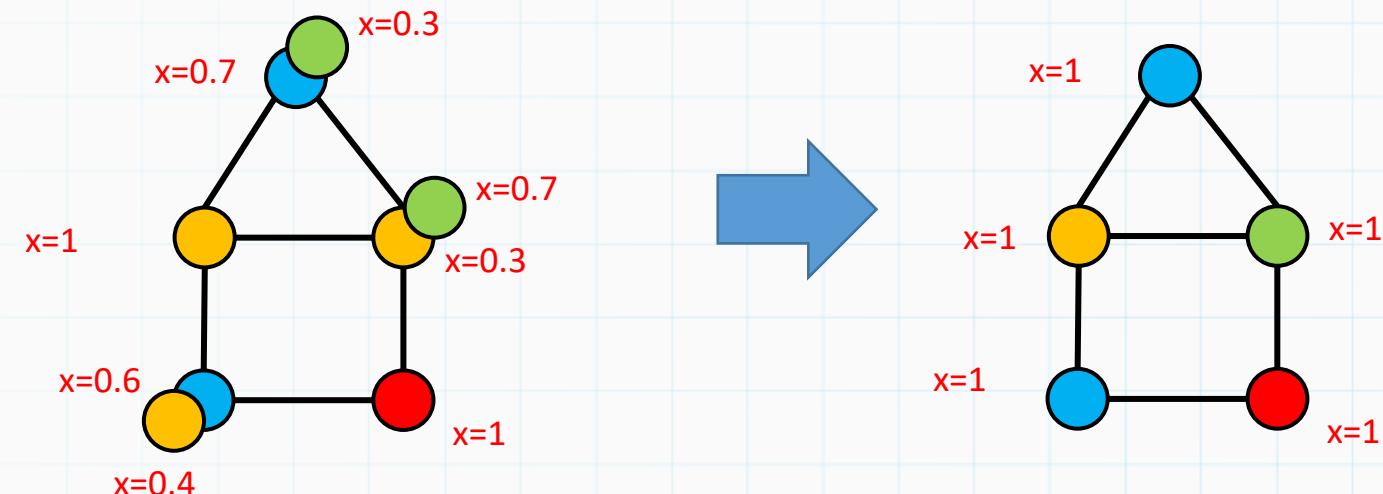


# CPLEX



E como as Heurísticas primais automáticas do CPLEX se comportam ?

- Chamada pelo CPLEX sempre que uma solução ótima de um subproblema é encontrada
- No nó raiz, é chamada após cada iteração da rotina dos cortes automáticos.
- O método devolve (se possível) para o CPLEX uma solução inteira derivada da solução relaxada (heurística linear)

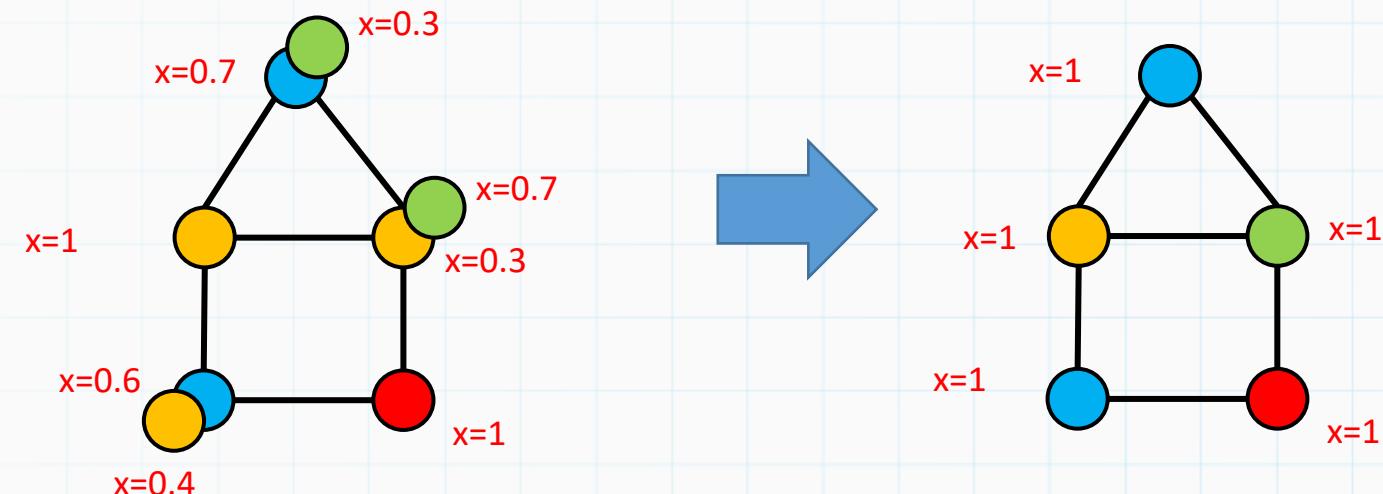


# CPLEX



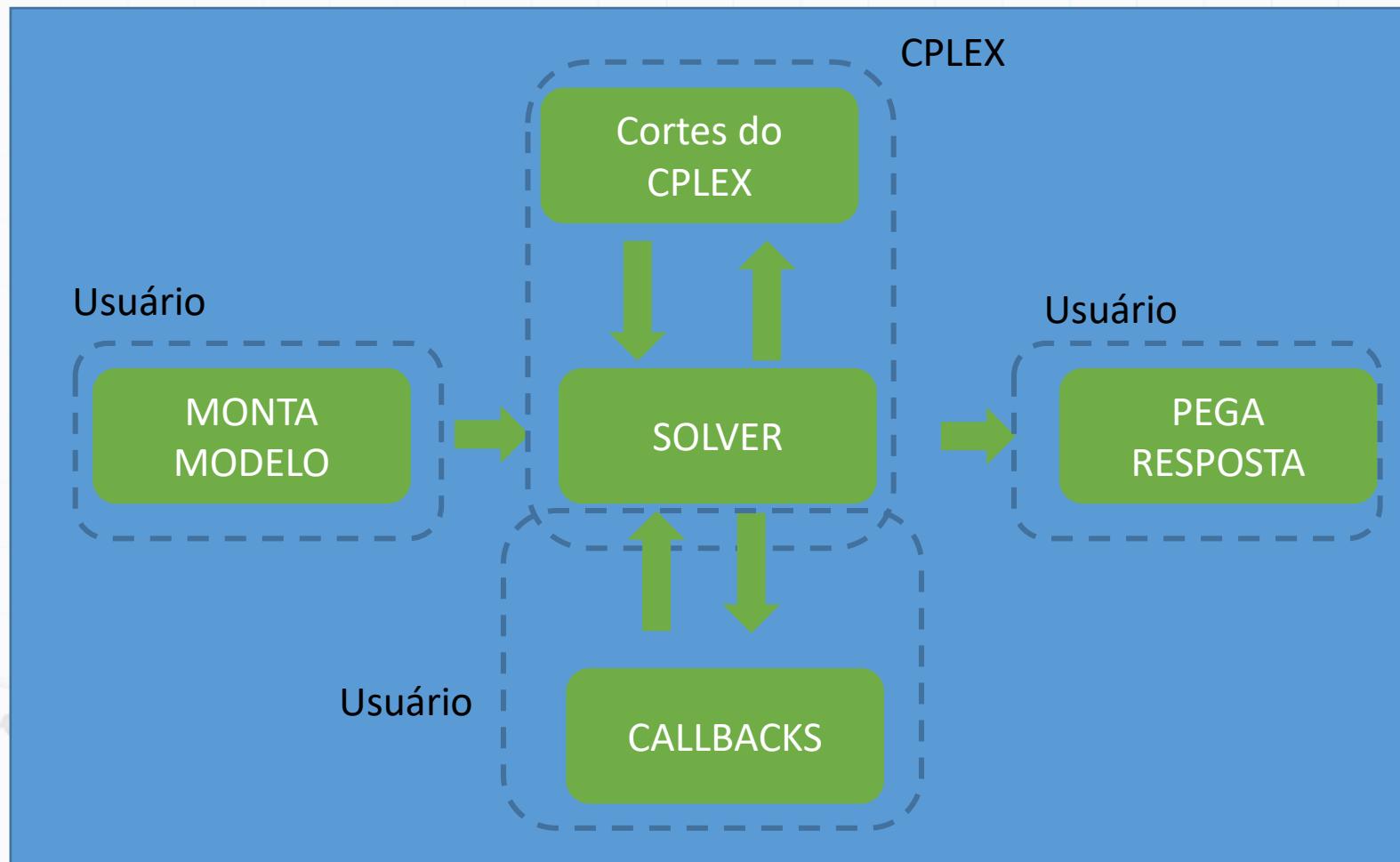
E como as Heurísticas primais automáticas do CPLEX se comportam ?

- Chamada pelo CPLEX sempre que uma solução ótima de um subproblema é encontrada
- No nó raiz, é chamada após cada iteração da rotina dos cortes automáticos.
- O método devolve (se possível) para o CPLEX uma solução inteira derivada da solução relaxada (heurística linear)
- Se uma melhor solução for achada, a melhor solução corrente é atualizada.



# Python-MIP

Funciona da mesma forma que no CPLEX:



Callbacks: Funções que interagem com o solver do CPLEX

# Python-MIP

TSP Assimétrico com MTZ

$$\text{MIN } \sum_{ij \in A} c_{ij} x_{ij}$$

sujeito a:

$$\sum_{j \in N^+(i)} x_{ij} = 1$$

$$\sum_{j \in N^-(i)} x_{ji} = 1$$

$$w_i + 1 \leq -w_j + M(1 - x_{ij}), \text{ para todo arco } ij \text{ em } A, \text{ com } i \neq j$$

$x_{ij} \in \{0,1\}$ , para todo arco  $ij$  pertencente a  $A$

$w_i$  número real, para todo vértice em  $V$

Vamos utilizar o exemplo que tínhamos e  
adicionar primeiro as restrições polinomiais  
do MTZ

# Python-MIP

TSP Assimétrico com MTZ

$$\text{MIN } \sum_{ij \in A} c_{ij} x_{ij}$$

sujeito a:

$$\sum_{j \in N^+(i)} x_{ij} = 1$$

$$\sum_{j \in N^-(i)} x_{ji} = 1$$

$$w_i + 1 \leq -w_j + M(1 - x_{ij}), \text{ para todo arco } ij \text{ em } A, \text{ com } i \neq j$$

$x_{ij} \in \{0,1\}$ , para todo arco  $ij$  pertencente a  $A$

$w_i$  número real, para todo vértice em  $V$

Vamos utilizar o exemplo que tínhamos e adicionar primeiro as restrições polinomiais do MTZ

```
# variaveis de eliminação de subciclo
w = [model.add_var(name='w_'+str(i), var_type=CONTINUOUS) for i in range(n)]

# restrições MTZ
M = n+1
for i in range(1,n):
    for j in range(1,n):
        if i != j:
            model.add_constr(w[i] + 1 <= w[j] + M - M*x[i][j], name='MTZ_'+str(i)+"_"+str(j))
```

Sol = 378 -> 547

# Python-MIP

TSP Assimétrico com MTZ

$$\text{MIN } \sum_{ij \in A} c_{ij} x_{ij}$$

sujeito a:

$$\sum_{j \in N^+(i)} x_{ij} = 1$$

$$\sum_{j \in N^-(i)} x_{ji} = 1$$

$$w_i + 1 \leq -w_j + M(1 - x_{ij}), \text{ para todo arco } ij \text{ em } A, \text{ com } i \neq j$$

$x_{ij} \in \{0,1\}$ , para todo arco  $ij$  pertencente a  $A$

$w_i$  número real, para todo vértice em  $V$



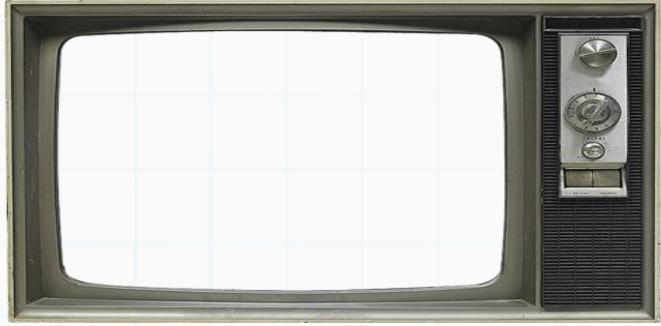
Com as restrições MTZ não teríamos necessidade de colocar cortes de usuários, mas vamos colocar também para acelerar o processo.

```
# variaveis de eliminação de subciclo
w = [model.add_var(name='w_'+str(i), var_type=CONTINUOUS) for i in range(n)]

# restrições MTZ
M = n+1
for i in range(1,n):
    for j in range(1,n):
        if i != j:
            model.add_constr(w[i] + 1 <= w[j] + M - M*x[i][j], name='MTZ_'+str(i)+"_"+str(j))
```

Sol = 378 -> 547

# Python-MIP



```
# passa classe de cortes de subciclo
model.cuts_generator = SubTourCutGenerator(x, n) ←
#model.lazy_constrs_generator = SubTourCutGenerator(x, n)

# otimiza
model.optimize(max_seconds=300) # limite de tempo
```

Passamos para o modelo, antes de otimizar, a referência da classe de **cortes de usuário**, que vamos criar, junto com todos os parâmetros que vamos precisar para identificar os cortes

```
from mip import *
import time
import networkx as nx # biblioteca de grafos ←
```

Vamos usar a biblioteca “**networkx**” para usar funções predefinidas para grafos. Teremos que instalar a biblioteca (repetir os mesmos passos que foram feitos para instalar a “mip”)

# Python-MIP

```
# gera cortes de subciclo
class SubTourCutGenerator(ConstrsGenerator): ←
    #construtor
    def __init__(self, x_, n_): ←
        self.x, self.n = x_, n_
    # função de identificação do corte
    #|(modelo, profundidade da arvore de BB, quantas passadas no nó ja fez)
    def generate_constrs(self, model: Model, depth: int = 0, npass: int = 0): ←
        x      = model.translate(self.x) # variáveis da sol. corrente
        cp    = CutPool()              # conjunto de cortes
        G     = nx.DiGraph()           # grafo auxiliar
        # constroi grafo residual ←
        for u in range(n):
            for v in range(n):
                if u != v:
                    G.add_edge(u, v, capacity=x[u][v].x)
```



# Python-MIP

```
# para cada par de vértices, acha o corte mínimo
for u in range(n):
    for v in range(n):
        if u != v:
            # função que encontra o corte mínimo se G for conexo
            # se grafo residual desconexo, o que ocorre em soluções inteiras
            # com subciclo, não vai achar o corte violado :(
            val, (S, NS) = nx.minimum_cut(G, u, v) ←

# encontra o corte mínimo de entrada em S (direcionado, por isso é 1 e não 2)
if val <= 0.99: ←
    soma = 0
    exp = 0
    for i in range(n):
        for j in range(n):
            if i != j and i in S and j in S:
                soma += x[i][j].x
                exp += x[i][j]
    if (soma > (len(S)-1) + 1e-4): ←
        cp.add(exp <= len(S)-1)
```

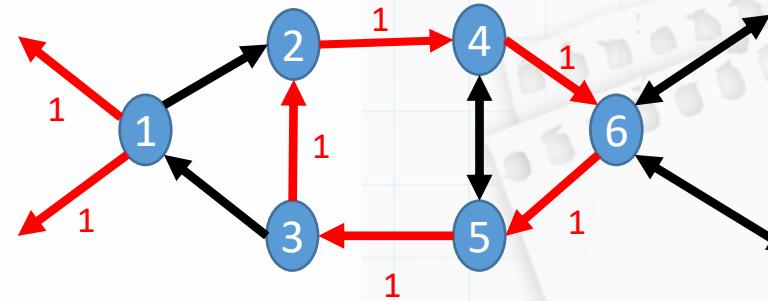
$$\sum_{\substack{i,j \in S \\ i \neq j}} x_{ij} \leq |S| - 1,$$

violado

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

Se grafo desconexo, retorna  
corte mínimo de  
componente conexa ☺

Não encontraria corte  
pois grafo desconexo



Mas como usamos MTZ, esses  
ciclos já foram eliminados

# Python-MIP

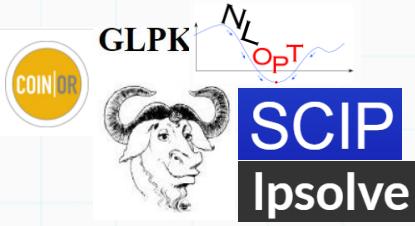
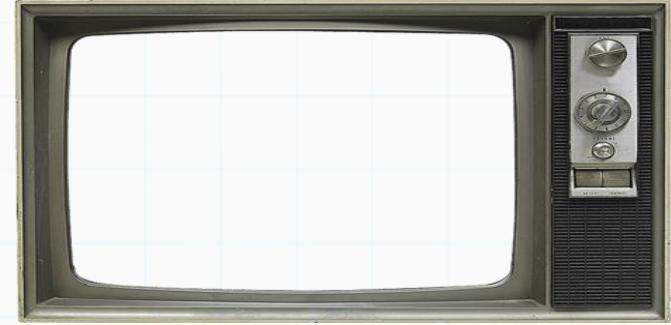
```
for cut in cp.cuts:  
    # insere corte de usuário  
    model.add_cut(cut)           ←  
    #model.add_lazy_constr(cut)  
    print(' ----- gerou corte de usuário -----')
```

```
melhor solução = 547.0  
limite inferior = 547.0  
tempo           = 1.5  
x_0_1  
x_1_4  
x_2_6  
x_3_12  
x_4_5  
x_5_13  
x_6_7  
x_7_8  
x_8_0  
x_9_11  
x_10_2  
x_11_3  
x_12_10  
x_13_9
```

# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

Depende



Modelo

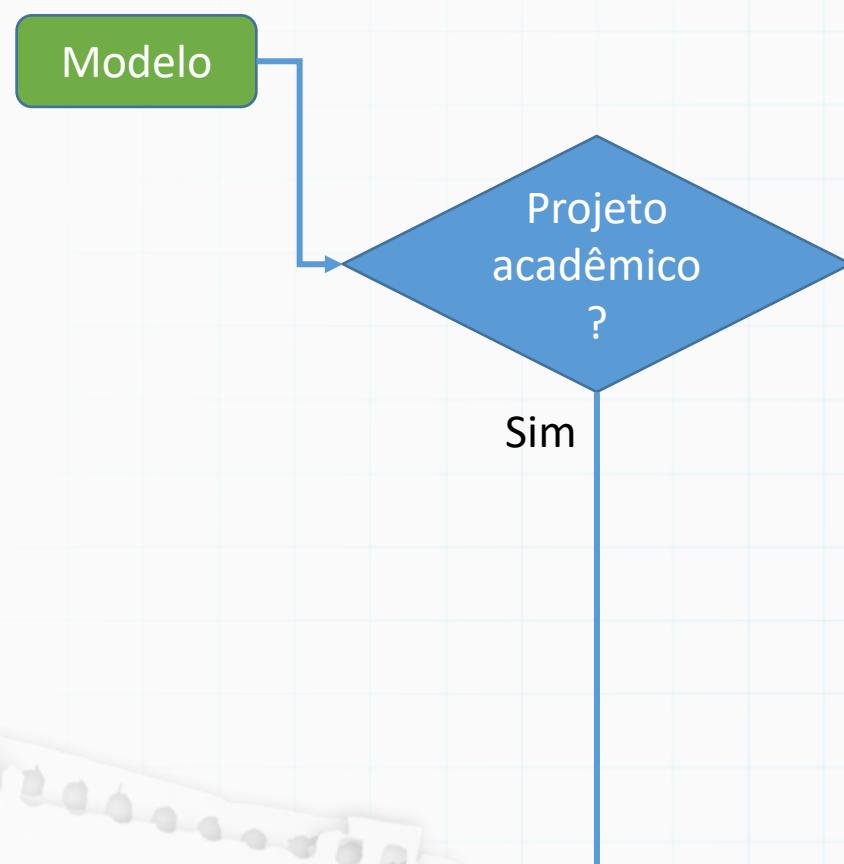
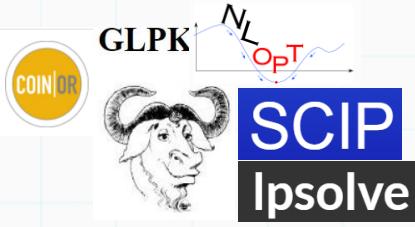
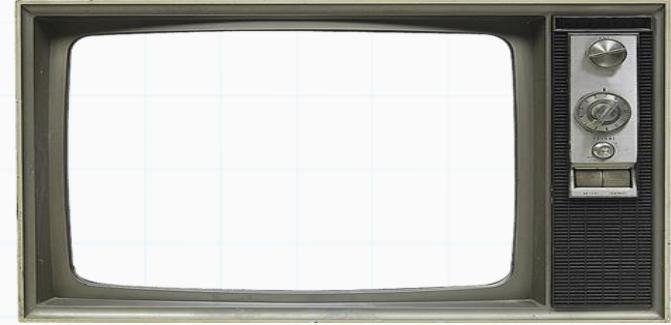
 GUROBI  
OPTIMIZATION

 IBM  ILOG  CPLEX  
 FICO  
Xpress

# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

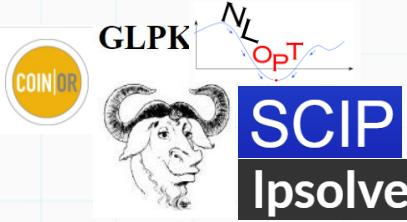
Depende



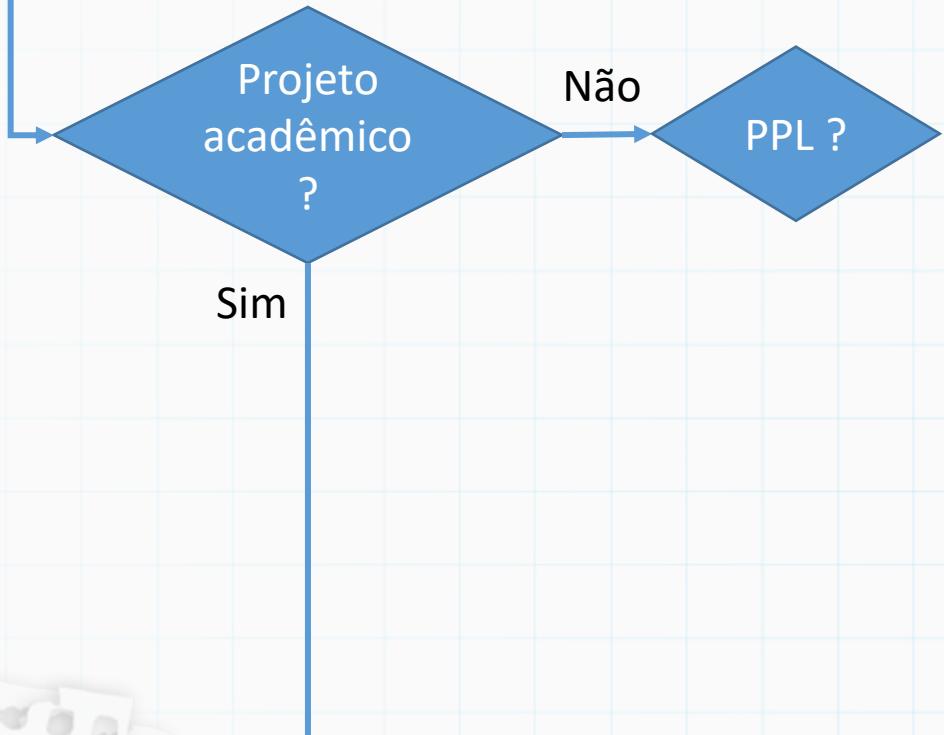
# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

Depende



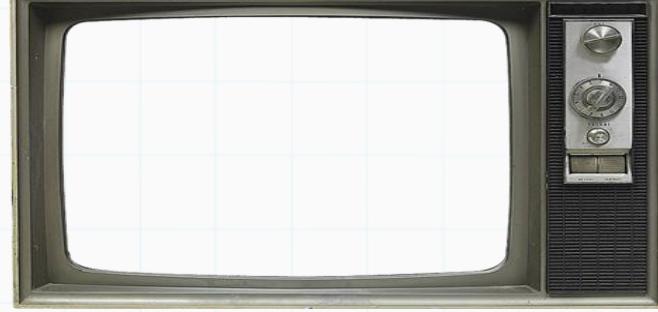
Modelo



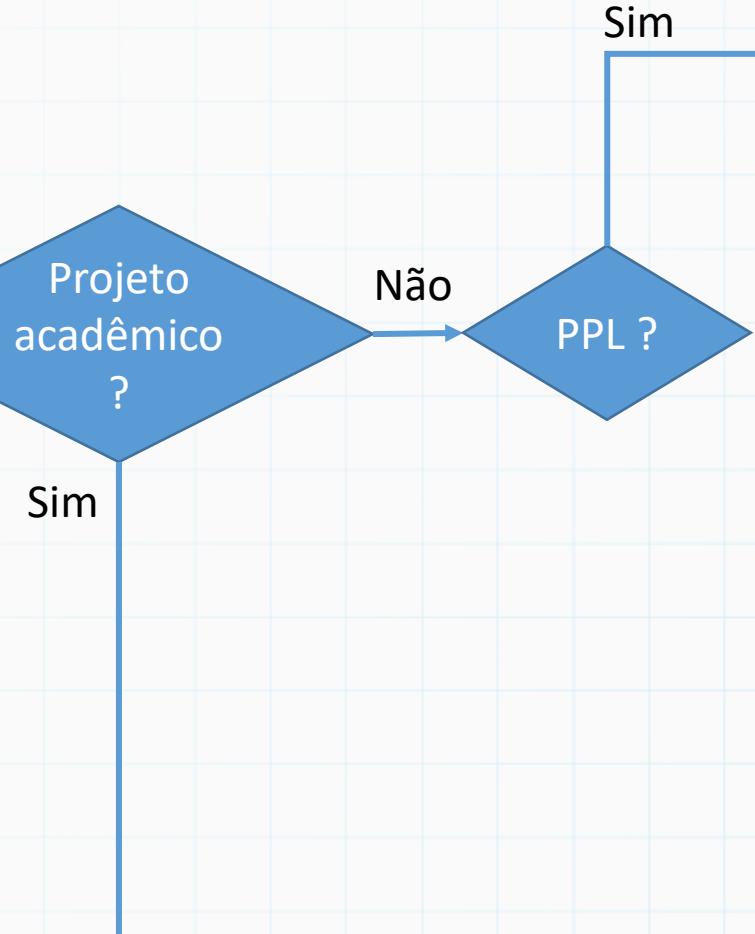
# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

Depende



Modelo



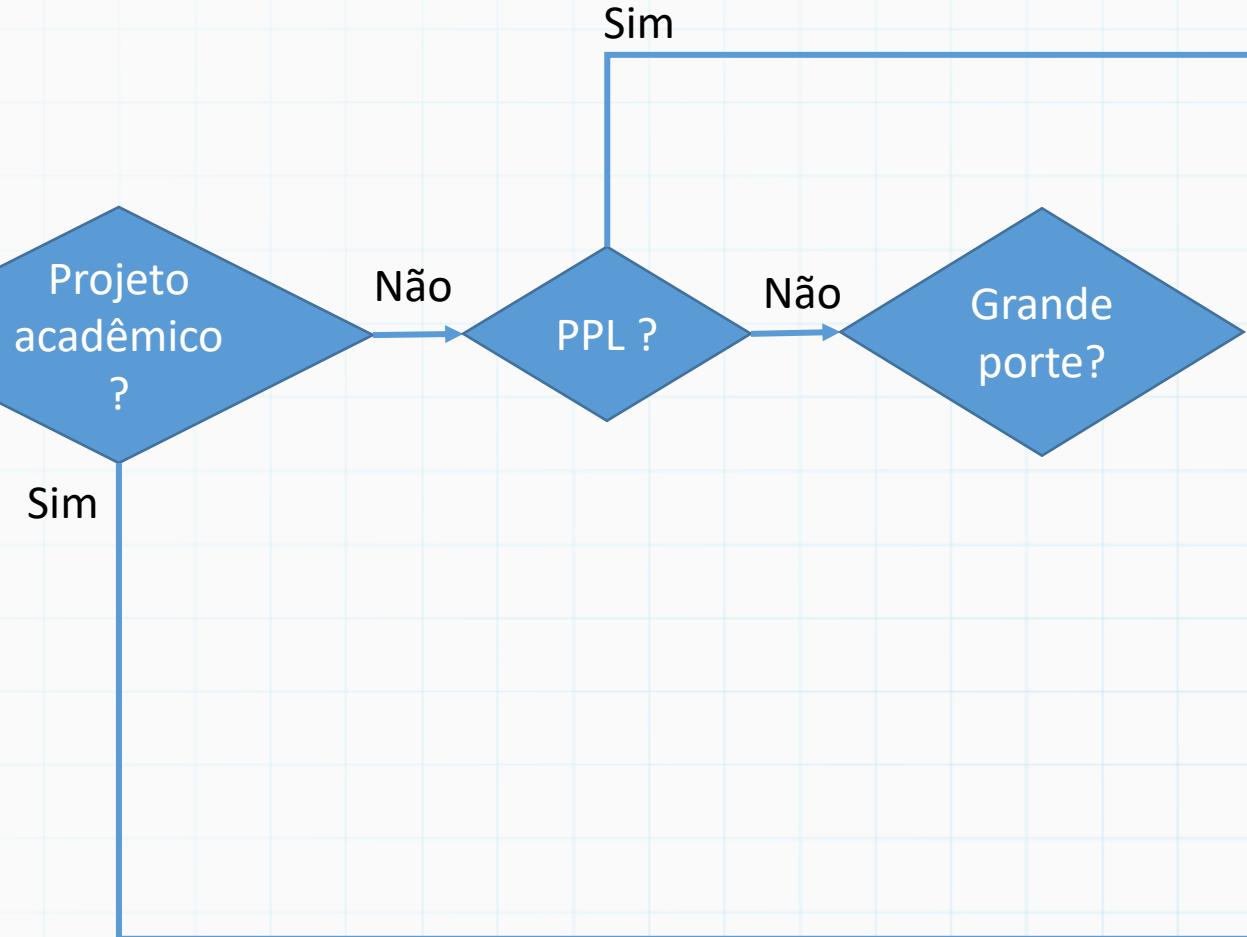
# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

Depende



Modelo



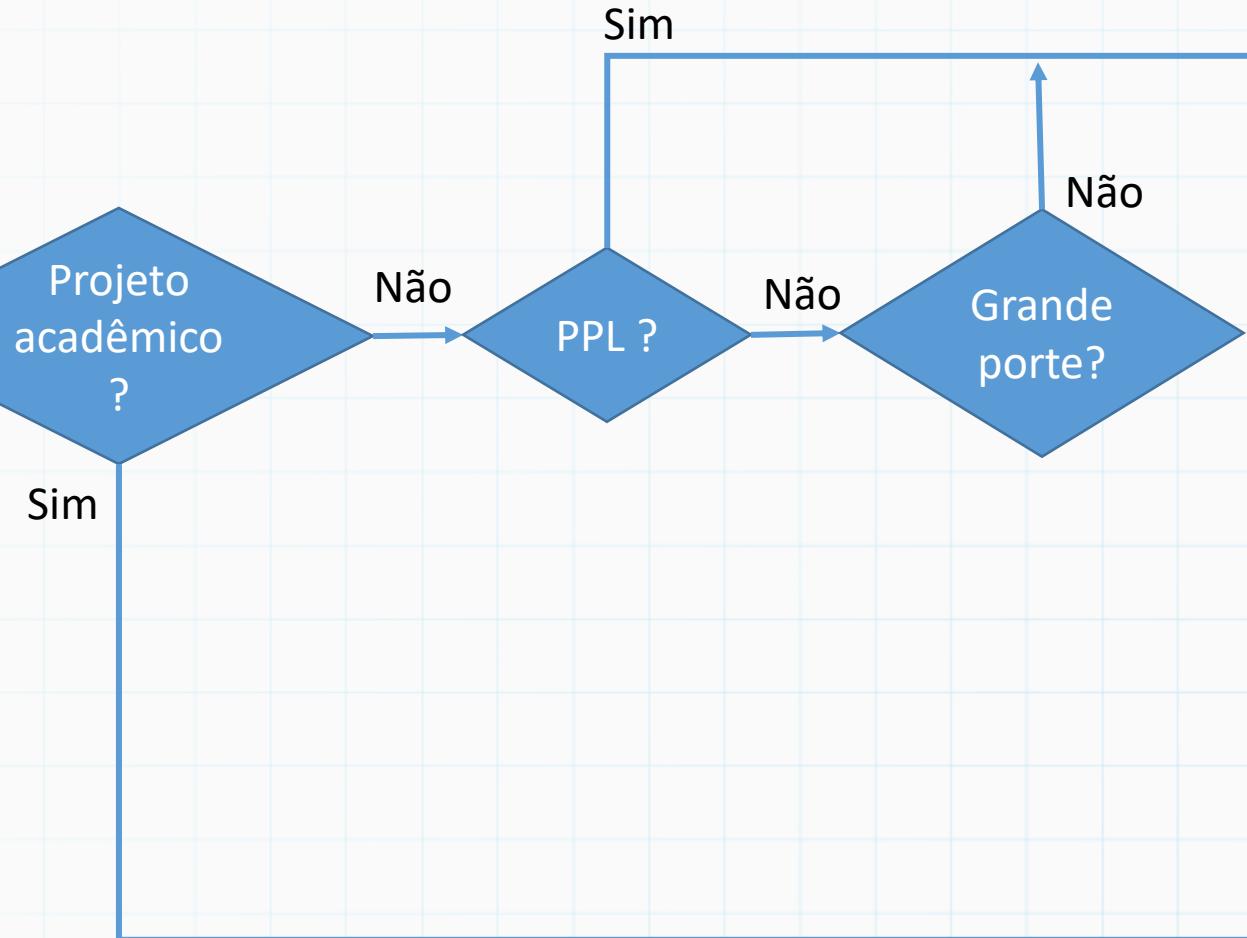
# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

Depende



Modelo

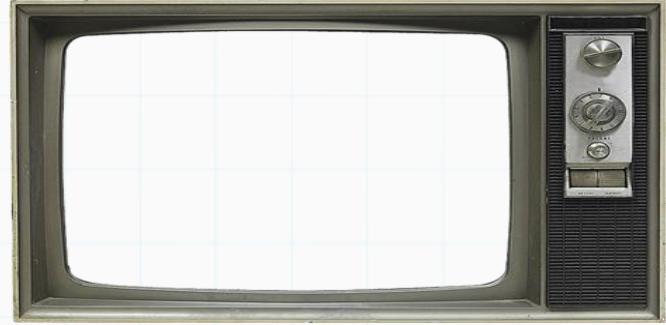


**CPLEX**

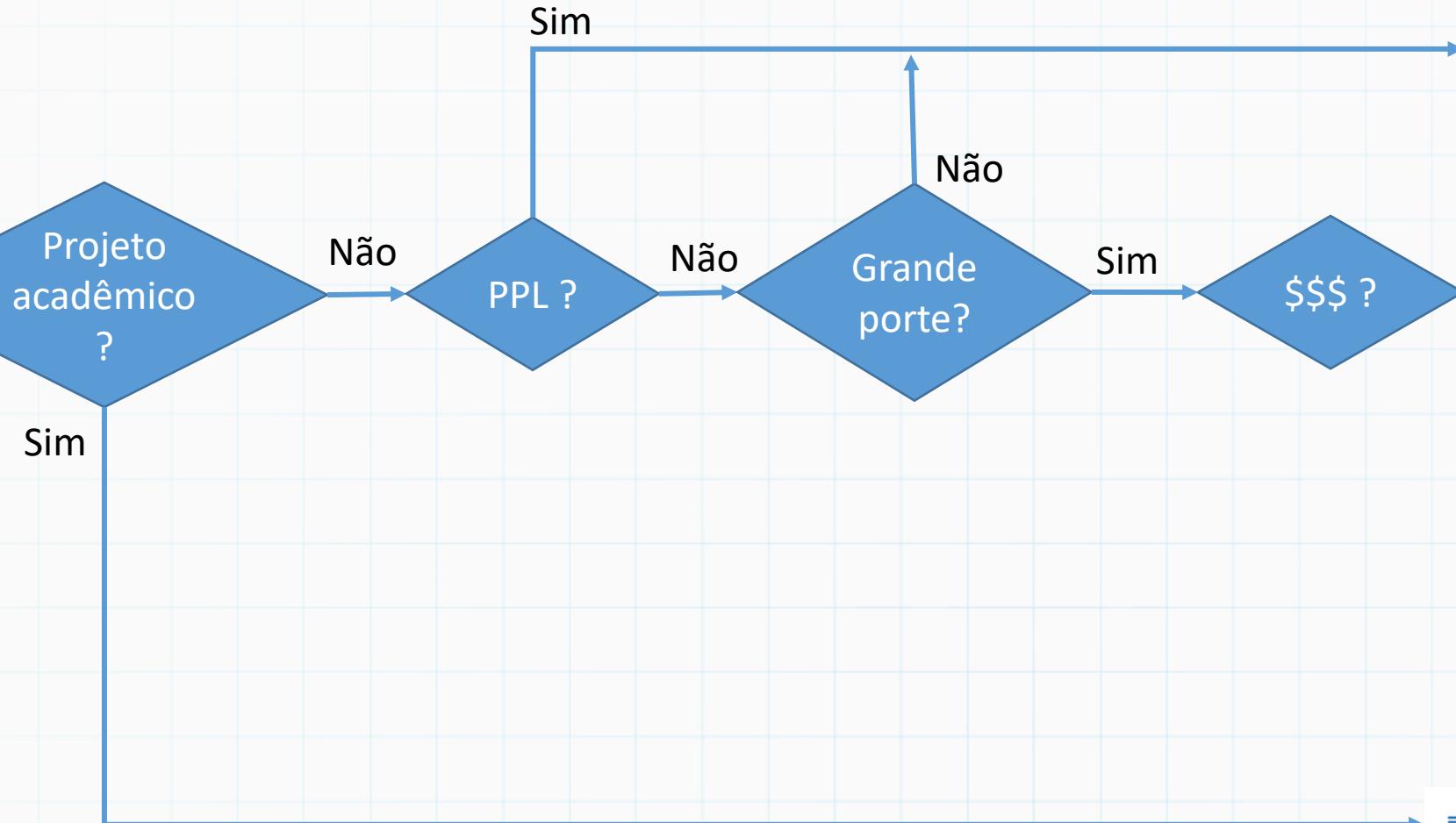
# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

Depende



Modelo



GUROBI  
OPTIMIZATION

IBM

ILOG

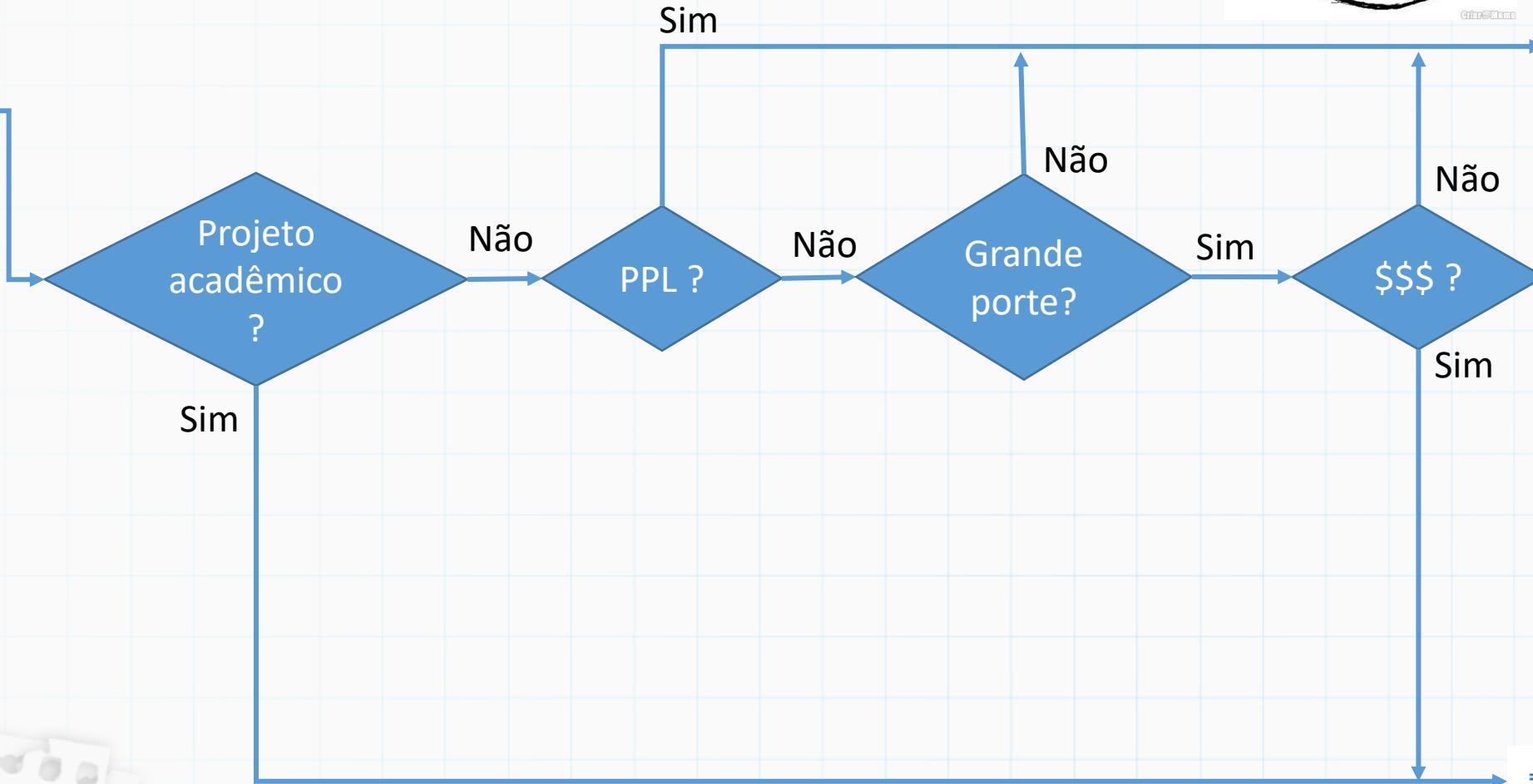
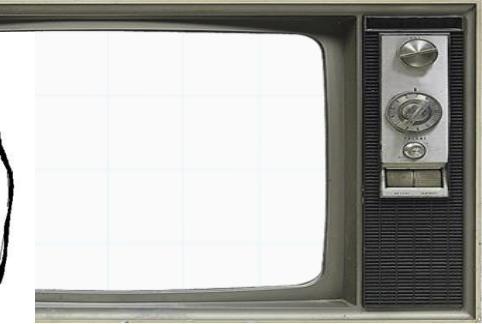
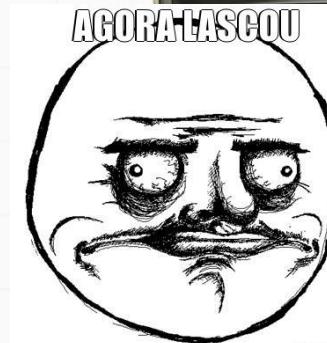
CPLEX

FICO  
Xpress

# CPLEX

Uau, então CPLEX é a melhor opção para resolver minha formulação ?

Depende



GUROBI  
OPTIMIZATION

IBM

ILOG

FICO  
Xpress

CPLEX

# CPLEX

Bacana, mas será que programação matemática é a melhor abordagem para meu problema ?

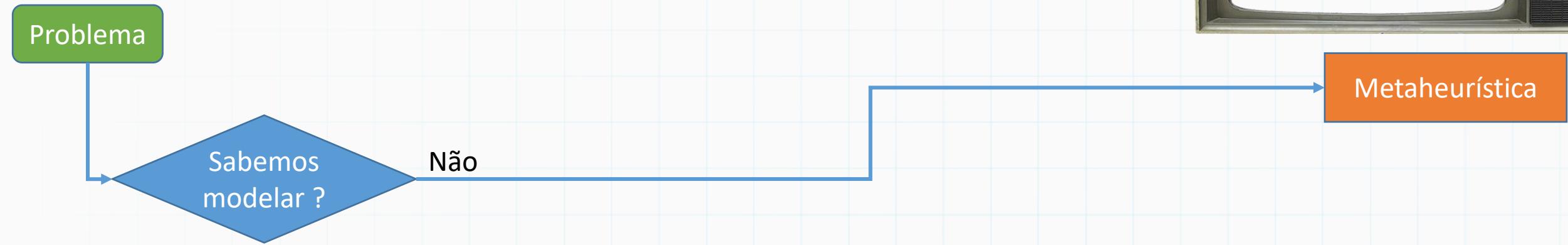
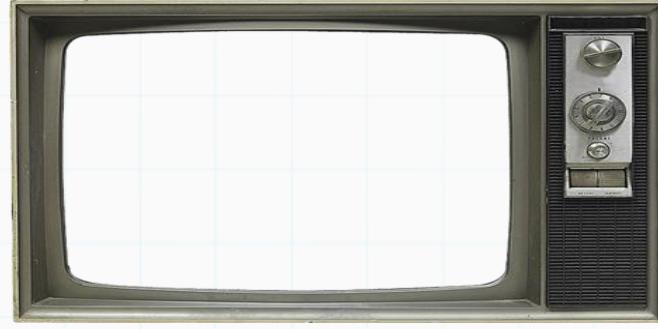
Problema

Metaheurística

Programação  
Matemática

# CPLEX

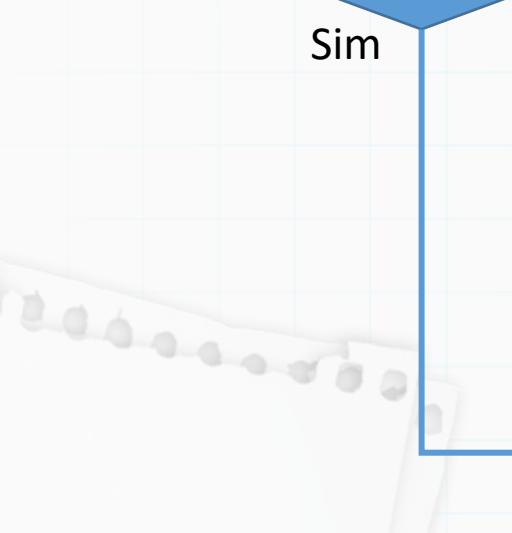
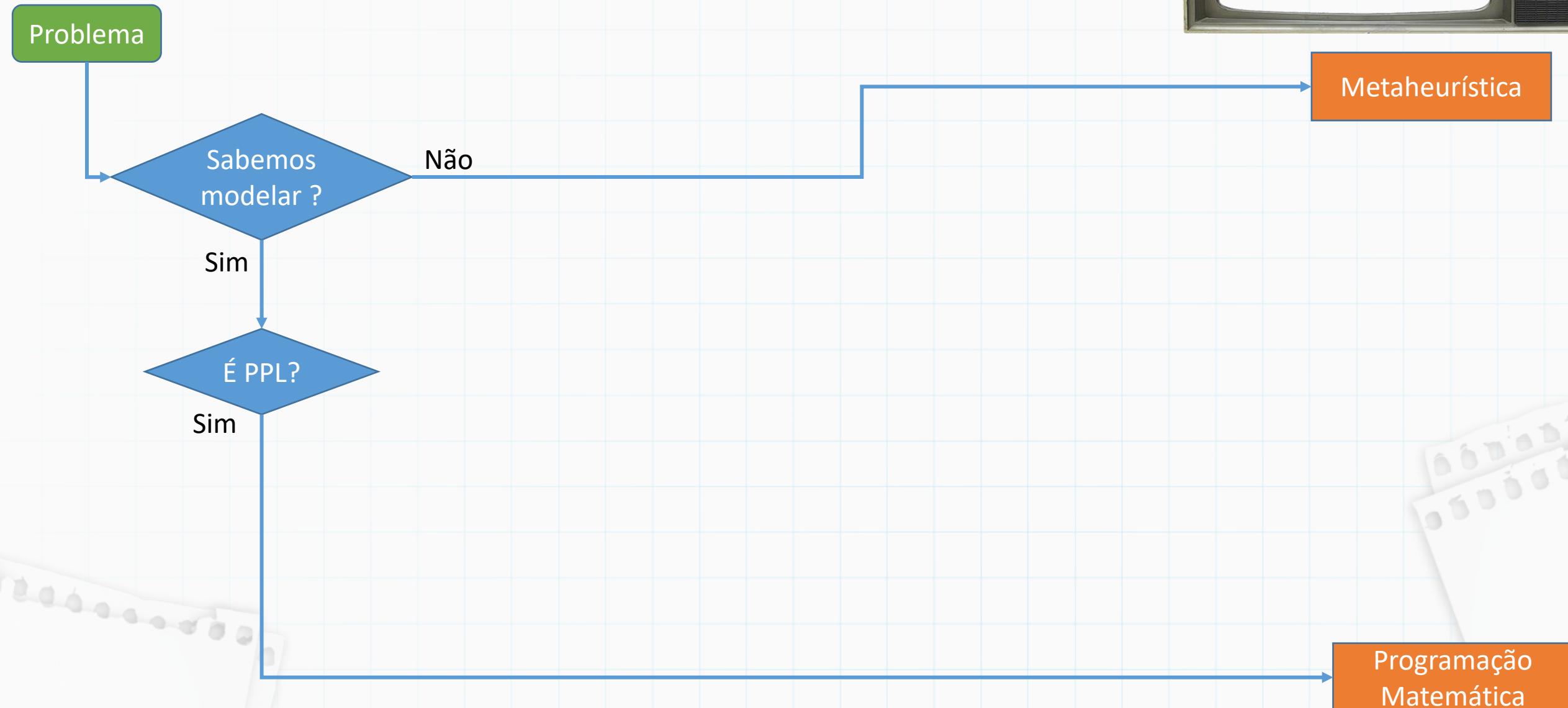
Bacana, mas será que programação matemática é a melhor abordagem para meu problema ?



Programação  
Matemática

# CPLEX

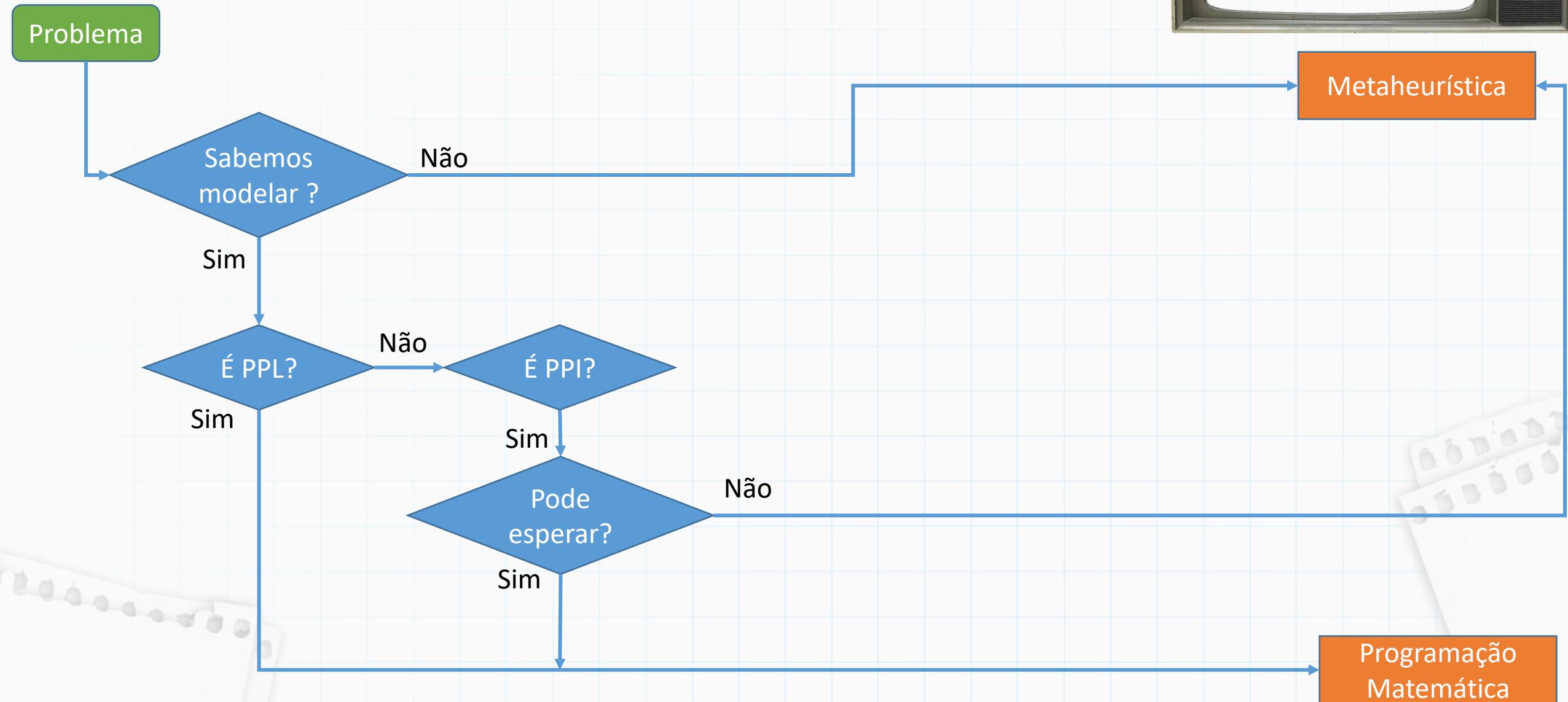
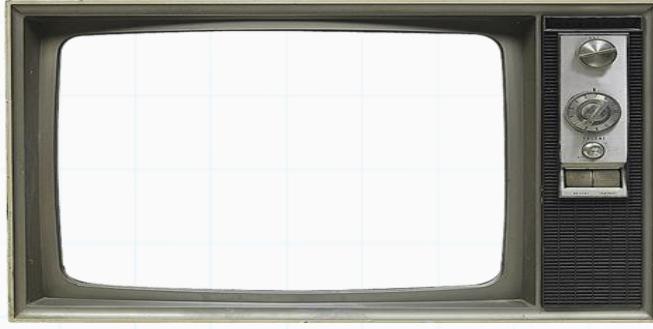
Bacana, mas será que programação matemática é a melhor abordagem para meu problema ?



Programação  
Matemática

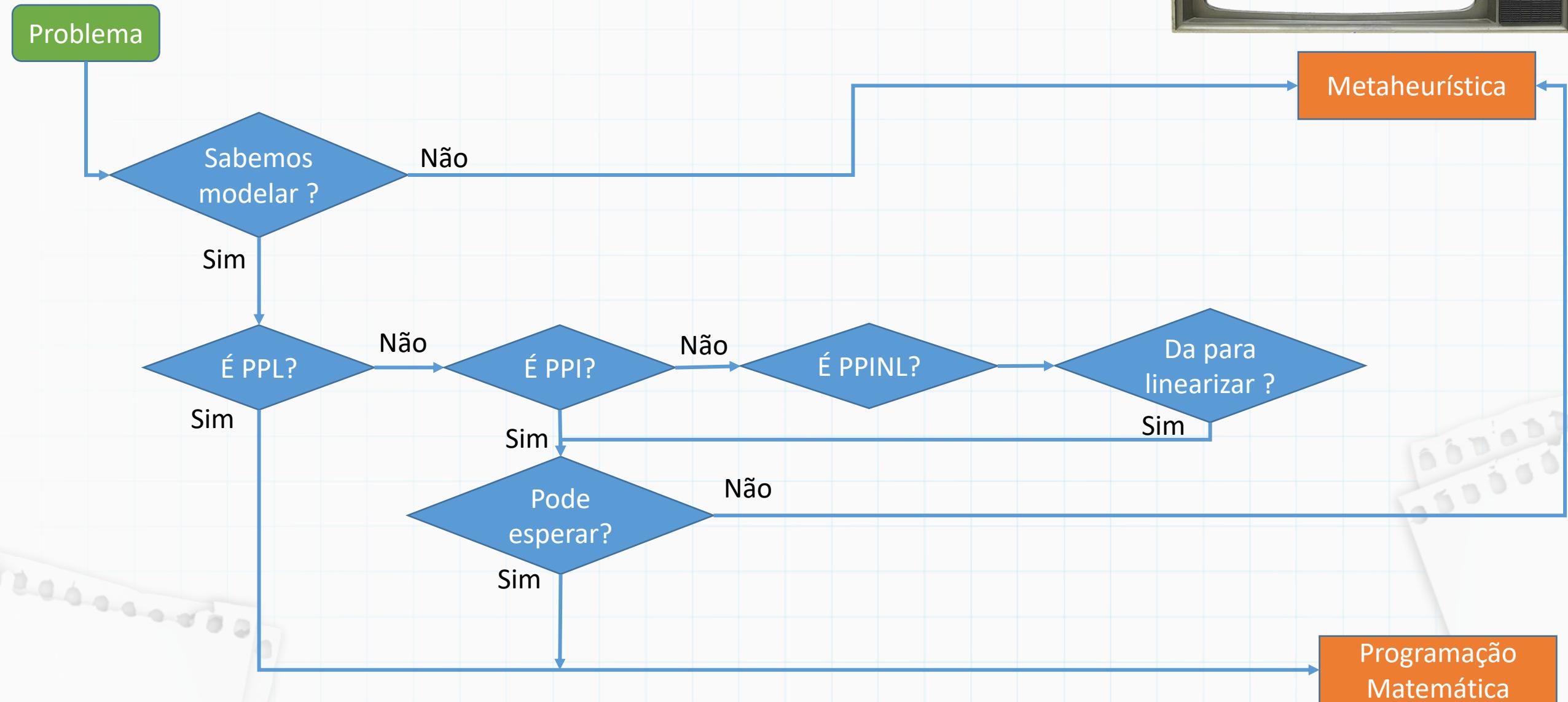
# CPLEX

Bacana, mas será que programação matemática é a melhor abordagem para meu problema ?



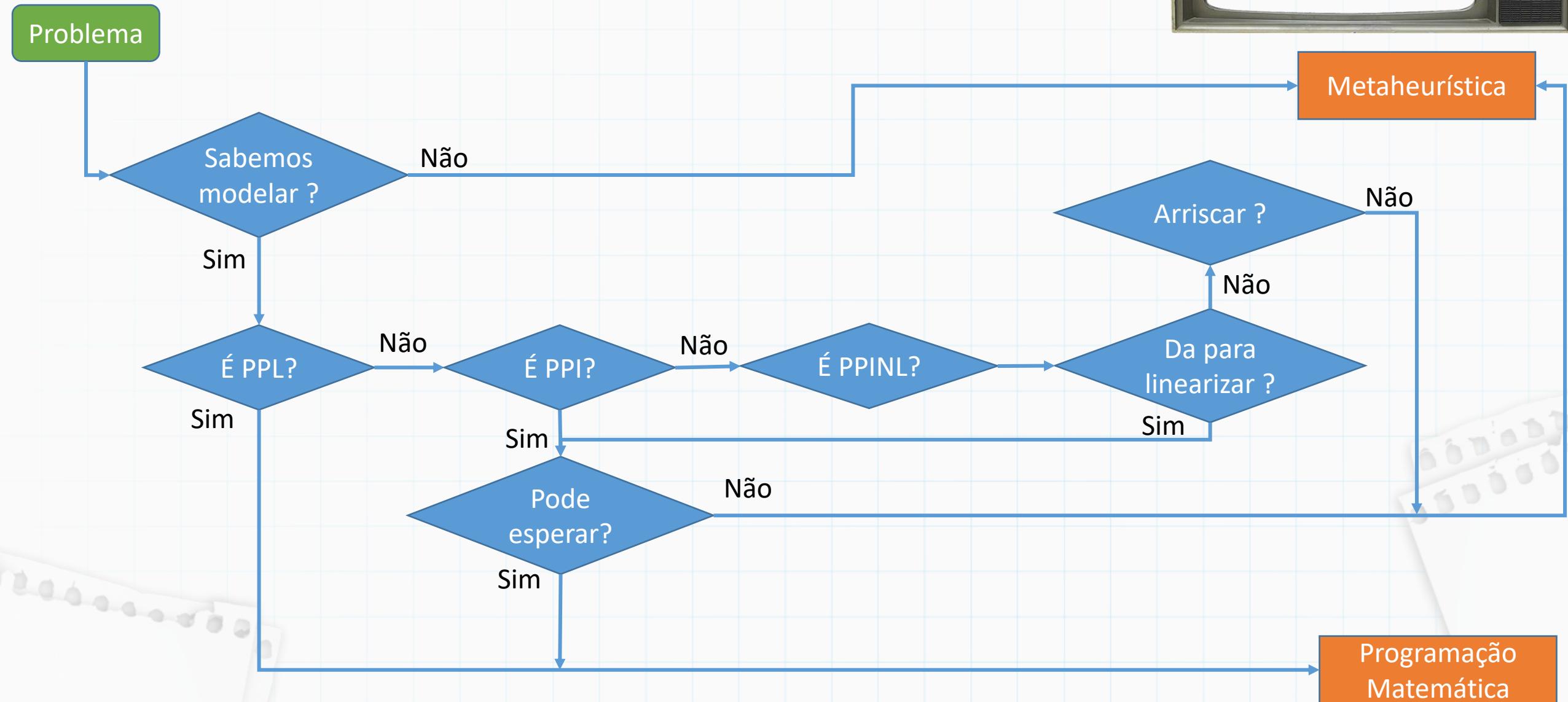
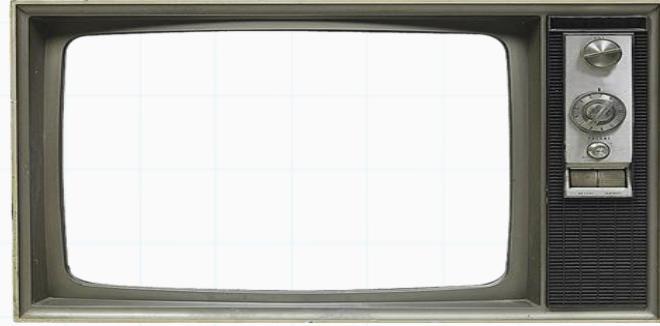
# CPLEX

Bacana, mas será que programação matemática é a melhor abordagem para meu problema ?



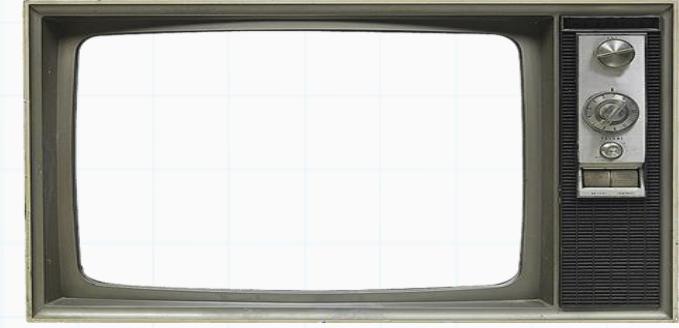
# CPLEX

Bacana, mas será que programação matemática é a melhor abordagem para meu problema ?

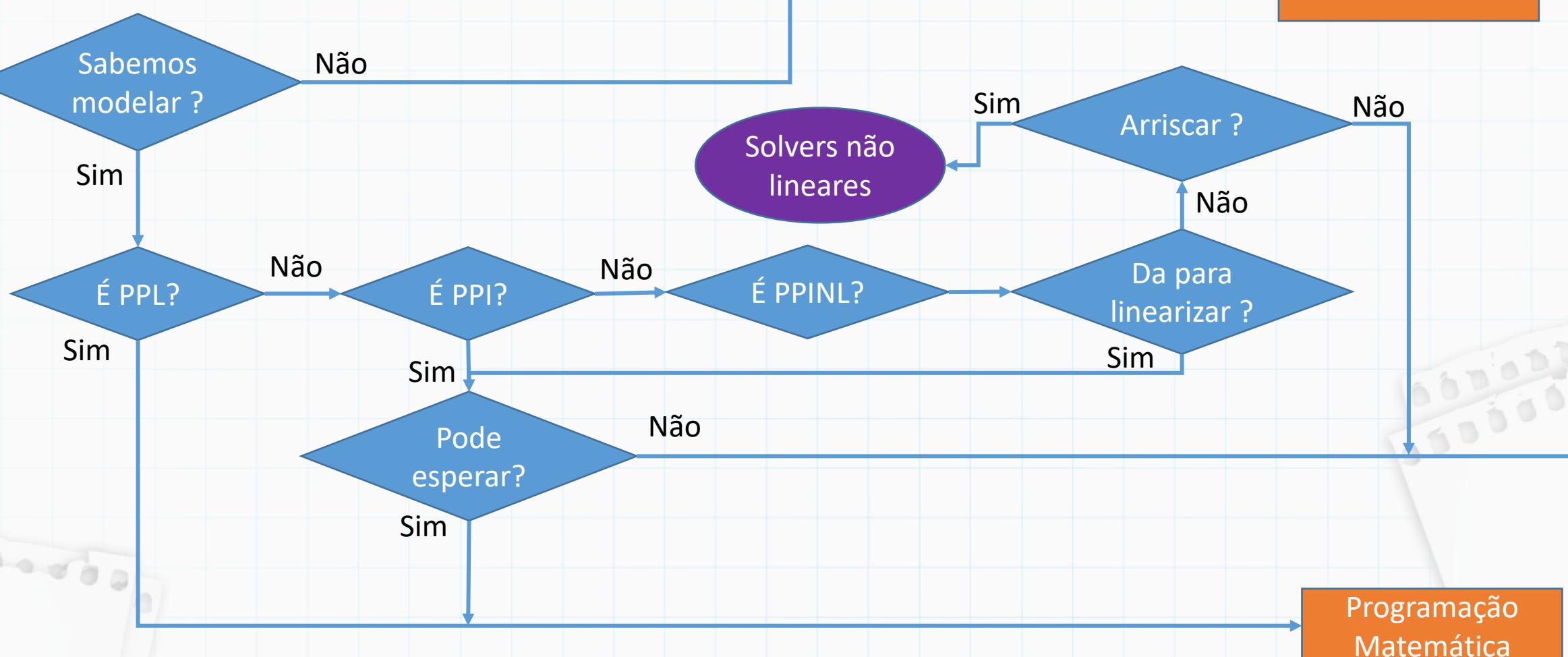


# CPLEX

Bacana, mas será que programação matemática é a melhor abordagem para o problema?

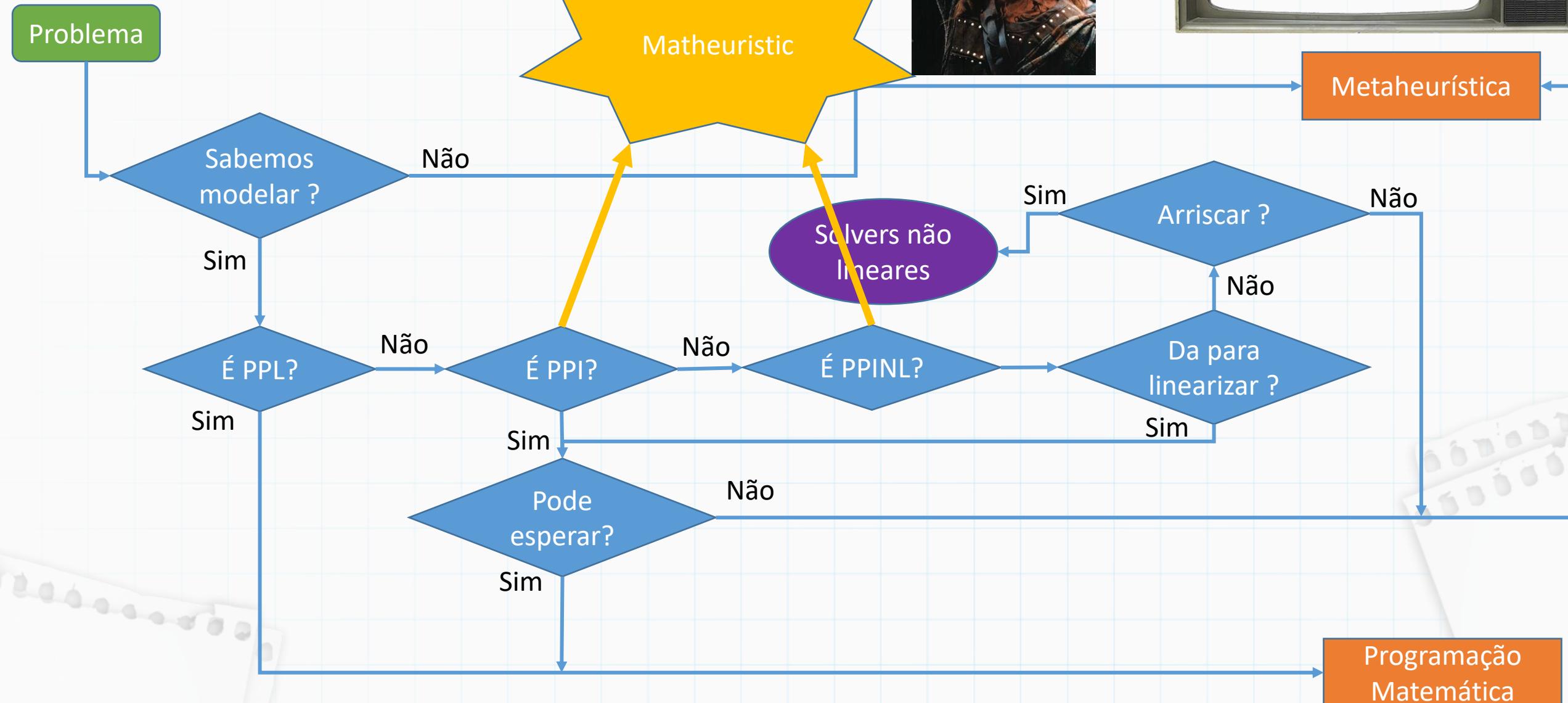
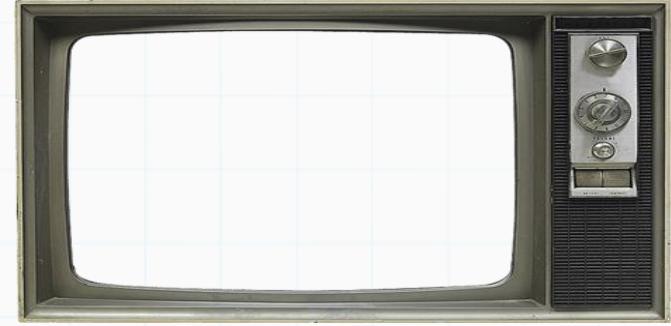


Problema



# CPLEX

Bacana, mas será que programação matemática é a melhor abordagem para o problema ?



# CPLEX



## Finalizando:

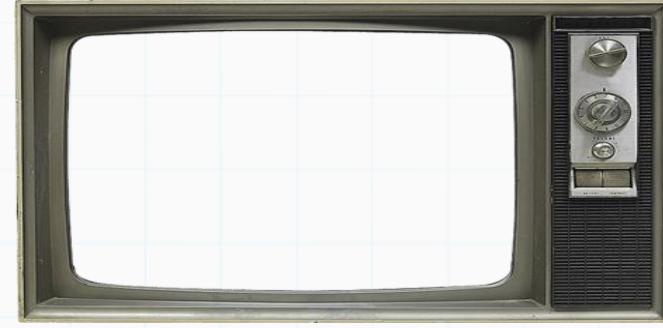
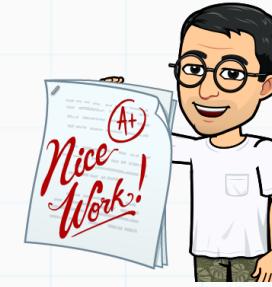
- Solvers são ferramentas poderosas de apoio para solução de problemas de decisão de forma exata e heurística (matheuristics)
  - Não existe solver que faça uma formulação ruim ter um bom desempenho
  - Também não adianta ter uma formulação forte e compacta executando num calhambeque



# Tarefa de Casa

CPLEX

- 1) baixar e executar o exemplo fornecido de branch-and-cut.
- 2) tente executar religando os cortes automáticos do CPLEX.
- 3) Mude os cortes de Lazy para User e teste.



```
solver.setParam(IloCplex::Param::Preprocessing::Presolve, 0);  
solver.setParam(IloCplex::Param::MIP::Limits::CutsFactor, 0);  
solver.setParam(IloCplex::Param::MIP::Strategy::HeuristicFreq, -1);
```

Python-MIP

- 1) baixar e executar o exemplo fornecido de branch-and-cut.
- 2) tente executar retirando a geração de cortes e deixando só o MTZ.

```
model.cuts_generator =
```

```
SubTourCutGenerator(x, n)
```

Até a próxima



