UFFLP: Integrando Programação Inteira Mista e Planilhas de Cálculo

Artur Pessoa

Eduardo Uchoa

Engenharia de Produção – UFF

Roteiro – Aula 1

- Apresentação geral da UFFLP
- Criação de modelos básicos com UFFLP
  - Exemplo 1: Problema do Mix de Produção
  - Exemplo 2: Problema da p-Mediana

Programação Linear e Inteira Mista

Importantes ferramentas básicas em Pesquisa Operacional

Pacotes resolvedores de PIM:

- Fechados (alto desempenho, licenças comerciais caras)
  - IBM ILOG CPLEX
  - FICO XPRESS
  - Gurobi (graças a ele, hoje todos esses pacotes oferecem licenças acadêmicas ilimitadas e gratuitas – sob certas condições).
- Abertos (médio desempenho)
  - COIN CBC
  - GLPK
  - MINLP
  -

Como usar esses pacotes?

1. Criar manualmente arquivos nos formatos MPS ou LP
   - Inviável exceto para modelos simples e poucos dados
   - Não separa o modelo dos dados
   - Muitos cursos de PO ainda não ensinam outros métodos => Muitos engenheiros de produção recém-formados não são capazes de aplicar PIM em problemas reais.

```
Maximize
 obj: 12 x1 + 5 x2 + 15 x3 + 10 x4
Subject To
 c1: 5 x1 + x2 + 9 x3 + 12 x4 <= 15
 c2: 3 x1 + 2 x2 + 4 x3 + 10 x4 <= 8
Bounds
 0 <= x1 <= 5
 0 <= x2 <= 5
 0 <= x3 <= 5
 0 <= x4 <= 5
Generals
 x1  x2  x3  x4
End
```

Como usar esses pacotes?

2. Usar interfaces integradas a planilhas (tipo Solver do Excel)
   - Sem dúvida melhor, mas ainda inviável para problemas reais com milhares de restrições.
   - Separação entre modelo e dados pouco clara.
   - Geralmente usam resolvedores com desempenho inferior.

Como usar esses pacotes?

3. Usar funções de programação C/C++/Java oferecidas pelos pacotes
   - Permite criar modelos de tamanho e complexidade arbitrários
   - “Baixo nível”, variáveis e restrições referenciadas pela numeração das colunas e linhas.
   - Exige conhecimento relativamente avançado de programação

```
int CPXaddcols (CPXENVptr env,
                CPXLPptr lp,
                int ccnt,
                int nzcnt,
                double *obj,
                int *cmatbeg,
                int *cmatind,
                double *cmatval,
                double *lb,
                double *ub,
                char **colname);
```

Como usar esses pacotes?

4. Usar linguagens de modelagem
   - Permite criar modelos de tamanho e complexidade arbitrários
   - “Alto nível”, variáveis e restrições representadas de forma semelhante a notação matemática
   - Separação entre modelo e dados
   - Exige pouco conhecimento de programação
     – Na verdade, modelos simples exigem um mínimo de programação; modelos complexos acabam exigindo bastante programação.

- Atualmente a melhor alternativa para um usuário que pretende fazer uso sério de PIM, mas não tem formação em computação.
- Particularmente apropriada para protótipos.

AMPL: diet1.mod

```
set NUTR ordered;
set FOOD ordered;

param cost {FOOD} >= 0;
param f_min {FOOD} >= 0, default 0;
param f_max {j in FOOD} >= f_min[j], default Infinity;

param n_min {NUTR} >= 0, default 0;
param n_max {i in NUTR} >= n_min[i], default Infinity;

param amt {NUTR,FOOD} >= 0;

var Buy {j in FOOD} integer >= f_min[j], <= f_max[j];

# ------------------------------------------------------

minimize Total_Cost:  sum {j in FOOD} cost[j] * Buy[j];

subject to Diet {i in NUTR}:
   n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```

AMPL: diet1.dat

```
param:  FOOD:                          cost  f_min  f_max :=
  "Quarter Pounder w/ Cheese"      1.84     .      .
  "McLean Deluxe w/ Cheese"        2.19     .      .
  "Big Mac"                        1.84     .      .
  "Filet-O-Fish"                   1.44     .      .
  "McGrilled Chicken"              2.29     .      .
  "Fries, small"                   0.77     .      .
  "Sausage McMuffin"               1.29     .      .
  "1% Lowfat Milk"                 0.60     .      .
  "Orange Juice"                   0.72     .      . ;

param:  NUTR:            n_min  n_max :=
        Cal              2000      .
        Carbo            350    375
        Protein          55      .
        VitA             100      .
        VitC             100      .
        Calc             100      .
        Iron             100      . ;
```

AMPL: diet1.dat

```
param amt (tr):
                                Cal  Carbo Protein  VitA  VitC  Calc  Iron :=
  "Quarter Pounder w/ Cheese"   510    34    28     15     6    30    20
  "McLean Deluxe w/ Cheese"     370    35    24     15    10    20    20
  "Big Mac"                     500    42    25      6     2    25    20
  "Filet-O-Fish"                370    38    14      2     0    15    10
  "McGrilled Chicken"           400    42    31      8    15    15     8
  "Fries, small"                220    26     3      0    15     0     2
  "Sausage McMuffin"            345    27    15      4     0    20    15
  "1% Lowfat Milk"              110    12     9     10     4    30     0
  "Orange Juice"                 80    20     1      2   120     2     2 ;
```

Pacotes que incluem linguagens de modelagem

- Fechados (incluem sofisticadas interfaces para importação/exportação de dados, ambiente de depuração, ferramentas de visualização, etc)
    - AMPL
        - Versão de estudante limitada a 300 var/rest
    - GAMS
        - Licença acadêmica c/ interface CPLEX: $1280
    - MOSEL (XPRESS)
        - Número limitado de licenças p/ Academic Partners
    - OPL (CPLEX)
        - Livre para Academic Initiative
- Abertos (praticamente só a própria linguagem)
    - COIN PuLP
    - Zimpl
    -

Uma crítica às linguagens de modelagem

Perfeitas para restrições simples, que só usam conceitos matemáticos previstos na linguagem (Ex: somatório de j=1 até m).

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i = 1, \ldots, n$$

Entretanto, restrições complexas obrigam o usuário a programar numa linguagem nova e relativamente pobre, que carece de algumas construções e recursos básicos encontradas em linguagens de uso geral ...

Uma crítica às linguagens de modelagem

Um usuário (mesmo que saiba programar!) tem uma grande dificuldade ao se deparar com a primeira restrição que usa um conceito matemático não diretamente suportado pela linguagem.

Ex: Uma enumeração de permutações, uma operação sobre grafos, etc.

UFFLP

Uma abordagem de “médio nível”:

- É um conjunto de funções em C/C++/VBA (em breve, Java)
- As variáveis e restrições são indexadas pelo **nome**.
- Outros conceitos matemáticos, simples ou complexos, são implementados na linguagem hospedeira.

UFFLP

- Modelos simples são escritos de forma quase tão simples quanto nas linguagem de modelagem.
  - Ex: um somatório de 1 a *m* exige um comando de iteração
    - em C: *for(j=1;j<=m;j++){...}*
    - *em VBA: For j=1 To n ... Next j*
- Modelos complexos se beneficiam de todas as construções e recursos existentes na linguagem hospedeira.
  - Ex: uma função recursiva para gerar permutações, uma bibilioteca de algoritmos sobre grafos.
  - Existe maior motivação para aprender algo que faz parte de uma linguagem de uso geral.
  - Farta documentação, exemplos na internet, etc.

UFFLP em VBA

O suporte à linguagem VBA (*Visual Basic for Applications*) é essencial na concepção da UFFLP:

- Apesar de injustamente criticada por puristas de computação, a linguagem Basic possui uma das mais rápidas curvas de aprendizado.

UFFLP em VBA

- A planilha Excel (que contém um interpretador VBA) fornece mais recursos do que os disponíveis em qualquer pacote contendo uma linguagem de modelagem:
  - Ambiente de depuração
  - Facilidade na manipulação de dados
  - Importação/Exportação para praticamente qualquer outra plataforma
  - Funções gráficas
  -

UFFLP em VBA

A UFFLP chamada de dentro do Excel vem sendo usada com sucesso desde 2007 nos cursos de pós-graduação em engenharia de produção da UFF:

- Alguns alunos que nunca tinham programado, ao final dos cursos se mostraram capazes de escrever aplicações de modelos de PIM para uso real.

UFFLP - Resolvedores de PIM

Atualmente a UFFLP suporta os resolvedores CPLEX e COIN CBC

Exemplo 1 – Problema do Mix de Produção

- Uma fábrica de cadeiras é capaz de produzir os seguintes modelos:

Exemplo 1 – Problema do Mix de Produção

- O limitante da produção é o fornecimento de 2 matéria-primas: lâminas de madeira e tecido

50 lâminas/semana

75 metros/semana

Lucro X gasto de matéria-prima

				
150	500	400	200	
1	4	3	1	
1	1	1	2	

Modelo de programação linear




$$\max \quad 150\ x_1 + 500\ x_2 + 400\ x_3 + 200\ x_4$$

$$1\ x_1 + 4\ x_2 + 3\ x_3 + 1\ x_4 \leq 50$$

$$1\ x_1 + 1\ x_2 + 1\ x_3 + 2\ x_4 \leq 75$$

Modelo de programação linear




$$\max \quad 150\ x_1 + 500\ x_2 + 400\ x_3 + 200\ x_4$$

$$1\ x_1 + 4\ x_2 + 3\ x_3 + 1\ x_4 \leq 50$$

$$1\ x_1 + 1\ x_2 + 1\ x_3 + 2\ x_4 \leq 75$$

Solução: $x_1 = 25$, $x_2 = 0$, $x_3 = 0$, $x_4 = 0$; lucro = R$8750,00

Funções básicas UFFLP

1. UFFLP_CreateProblem
2. UFFLP_AddVariable
3. UFFLP_SetCoefficient
4. UFFLP_AddConstraint
5. UFFLP_Solve
6. UFFLP_GetObjValue
7. UFFLP_GetSolution
8. UFFLP_DestroyProblem

Usadas em praticamente qualquer aplicação

Outras funções UFFLP usadas no exemplo

- UFFLP_WriteLP
- UFFLP_SetLogInfo

UFFLP: linguagem C

```c
/*****************************************************************************
*             Problema do Mix de Produção (instância fixa)
*****************************************************************************/

#include "UFFLP.h"

int main(void)
{
    // Declara variáveis do programa
    UFFProblem* problema;
    int result, status;
    double lucro, x1, x2, x3, x4;

    // Cria o problema
    problema = UFFLP_CreateProblem(UFFLP_Maximize);

    // Cria variáveis da formulação
    result = UFFLP_AddVariable(problema, "x1", 0, UFFLP_Infinity, 150, UFFLP_Continuous);
    result = UFFLP_AddVariable(problema, "x2", 0, UFFLP_Infinity, 400, UFFLP_Continuous);
    result = UFFLP_AddVariable(problema, "x3", 0, UFFLP_Infinity, 300, UFFLP_Continuous);
    result = UFFLP_AddVariable(problema, "x4", 0, UFFLP_Infinity, 200, UFFLP_Continuous);

    // Cria a restrição de disponibilidade de madeira
    result = UFFLP_SetCoefficient(problema, "madeira", "x1", 1);
    result = UFFLP_SetCoefficient(problema, "madeira", "x2", 4);
    result = UFFLP_SetCoefficient(problema, "madeira", "x3", 3);
    result = UFFLP_SetCoefficient(problema, "madeira", "x4", 1);
    result = UFFLP_AddConstraint(problema, "madeira", 50, UFFLP_Less);
```

UFFLP: linguagem C

```
29
30          // Cria a restrição de disponibilidade de tecido
31          result = UFFLP_SetCoefficient(problema, "tecido", "x1", 1);
32          result = UFFLP_SetCoefficient(problema, "tecido", "x2", 1);
33          result = UFFLP_SetCoefficient(problema, "tecido", "x3", 1);
34          result = UFFLP_SetCoefficient(problema, "tecido", "x4", 2);
35          result = UFFLP_AddConstraint(problema, "tecido", 75, UFFLP_Less);
36
37          // Configura o arquivo de log
38          result = UFFLP_SetLogInfo(problema, "mix.log", 2);
39
40          // Escreve o modelo num arquivo LP
41          result = UFFLP_WriteLP(problema, "mix.lp");
42
43          // Resolve o problema
44          status = UFFLP_Solve(problema);
45
46          // Pega a solução
47          result = UFFLP_GetObjValue(problema, &lucro);
48          result = UFFLP_GetSolution(problema, "x1", &x1);
49          result = UFFLP_GetSolution(problema, "x2", &x2);
50          result = UFFLP_GetSolution(problema, "x3", &x3);
51          result = UFFLP_GetSolution(problema, "x4", &x4);
52
53      return 0;
54  }
```

UFFLP: linguagem VBA

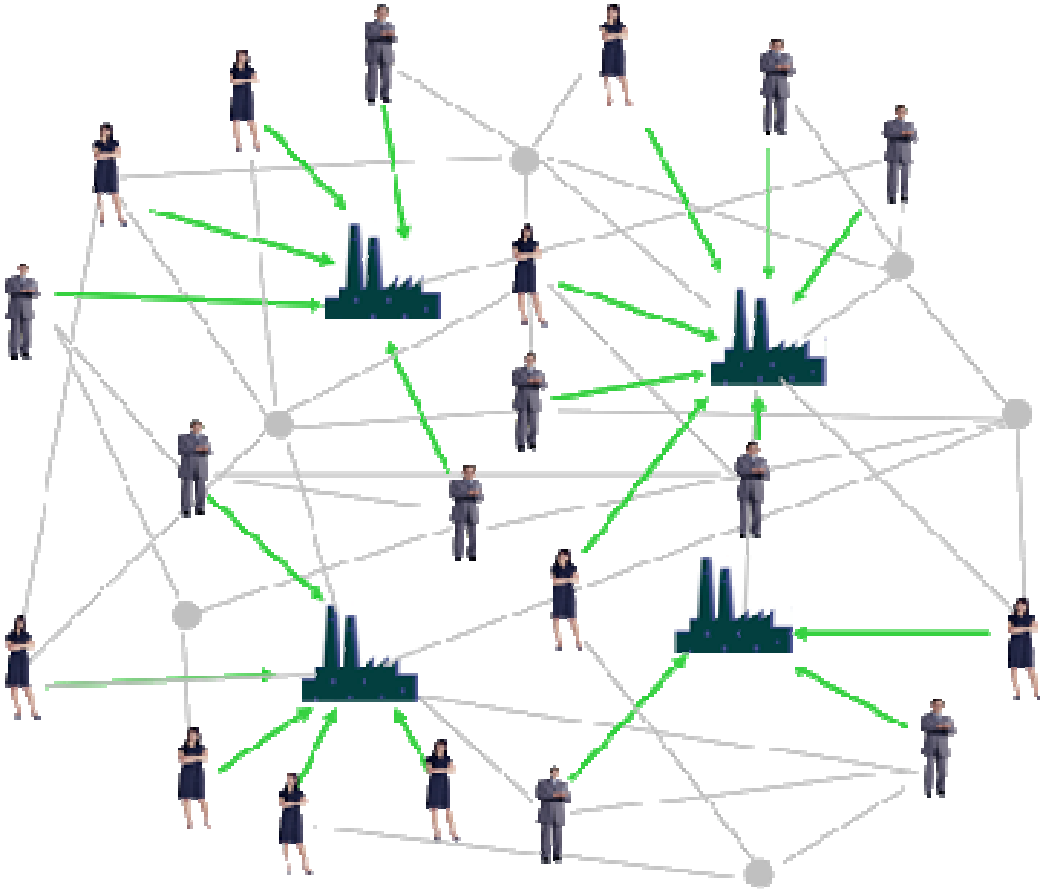
```
'*************************************************************************
'*              Problema do Mix de Produção (instância fixa)
'*************************************************************************

Sub Otimiza()
    ' Declara variáveis do programa
    Dim problema As Long, result As Long, status As Long
    Dim lucro As Double, x1 As Double, x2 As Double, x3 As Double, x4 As Double

    ' Cria o problema
    problema = UFFLP_CreateProblem(UFFLP_Maximize)

    ' Cria variáveis da formulação
    result = UFFLP_AddVariable_(problema, "x1", 0, UFFLP_Infinity, 150, UFFLP_Continuous)
    result = UFFLP_AddVariable_(problema, "x2", 0, UFFLP_Infinity, 400, UFFLP_Continuous)
    result = UFFLP_AddVariable_(problema, "x3", 0, UFFLP_Infinity, 300, UFFLP_Continuous)
    result = UFFLP_AddVariable_(problema, "x4", 0, UFFLP_Infinity, 200, UFFLP_Continuous)

    ' Cria a restrição de disponibilidade de madeira
    result = UFFLP_SetCoefficient_(problema, "madeira", "x1", 1)
    result = UFFLP_SetCoefficient_(problema, "madeira", "x2", 4)
    result = UFFLP_SetCoefficient_(problema, "madeira", "x3", 3)
    result = UFFLP_SetCoefficient_(problema, "madeira", "x4", 1)
    result = UFFLP_AddConstraint_(problema, "madeira", 50, UFFLP_Less)
```

UFFLP: linguagem VBA

```
    ' Cria a restrição de disponibilidade de tecido
    result = UFFLP_SetCoefficient_(problema, "tecido", "x1", 1)
    result = UFFLP_SetCoefficient_(problema, "tecido", "x2", 1)
    result = UFFLP_SetCoefficient_(problema, "tecido", "x3", 1)
    result = UFFLP_SetCoefficient_(problema, "tecido", "x4", 2)
    result = UFFLP_AddConstraint_(problema, "tecido", 75, UFFLP_Less)

    ' Configura o arquivo de log
    result = UFFLP_SetLogInfo_(problema, ThisWorkbook.Path & "/" & "mix.log", 2)

    ' Escreve o modelo num arquivo LP
    result = UFFLP_WriteLP_(problema, ThisWorkbook.Path & "/" & "mix.lp")

    ' Resolve o problema
    status = UFFLP_Solve(problema)

    ' Pega a solução
    result = UFFLP_GetObjValue_(problema, lucro)
    result = UFFLP_GetSolution_(problema, "x1", x1)
    result = UFFLP_GetSolution_(problema, "x2", x2)
    result = UFFLP_GetSolution_(problema, "x3", x3)
    result = UFFLP_GetSolution_(problema, "x4", x4)
End Sub
```

Exemplo 2 – Problema das *p*-medianas

- *n* clientes
- *m* locais potenciais p/ abrir algum serviço
- Distâncias d_{ij} entre cliente *i* e local *j*
- Escolher *p* locais para minimizar a soma das distâncias de cada cliente ao local aberto mais próximo.



Exemplo 2 – Problema das *p*-medianas

Variáveis:

- x_j (j = 1, ..., m) = 1 se o local j é aberto
- y_{ij} (i = 1, ..., n; j = 1, ..., m) = 1 se o cliente i é atendido no local j

Exemplo 2 – Problema das *p*-medianas

$$\text{Min} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} d_{ij} y_{ij}$$

$$\text{S.a} \quad \sum_{j=1}^{m} y_{ij} = 1 \qquad \forall i = 1,\ldots n$$

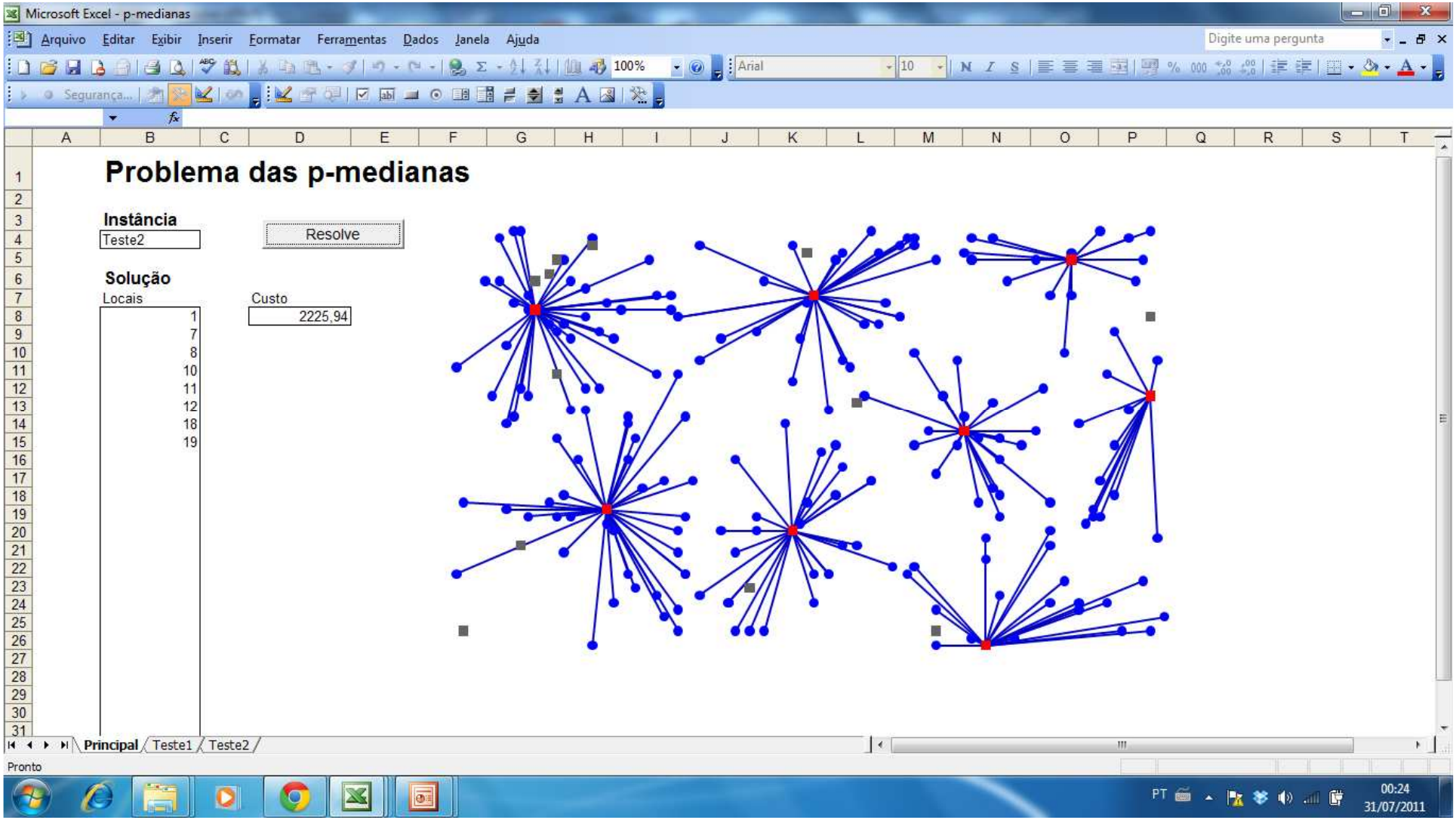
$$y_{ij} - x_j \leq 0 \qquad \forall i = 1,\ldots n; \forall j = 1,\ldots m$$

$$\sum_{j=1}^{m} x_j = p$$

$$y_{ij} \in \{0,1\} \qquad \forall i = 1,\ldots n; \forall j = 1,\ldots m$$

$$x_j \in \{0,1\} \qquad \forall j = 1,\ldots m$$

Tela principal Excel



Criação das variáveis

```
' Cria uma instância de MIP na UFFLP
Dim problema As Long
problema = UFFLP_CreateProblem(UFFLP_Minimize)

'----------------- Cria as variáveis -----------------

' Função objetivo: "Minimizar Soma(i=1..n) Soma(j=1..m) dist(i,j) y_i_j"
'     onde "dist(i,j)" é a distância euclidiana do cliente i ao local j
' Variáveis "y_i_j": binárias onde 1 indica que o cliente i é atendida no local j
Dim variavel As String
Dim erro As Long
Dim dist As Double
For i = 1 To n
    For j = 1 To m
        ' calcula dist(i,j)
        dist = Sqr((Clientes(i, 1) - Locais(j, 1)) ^ 2 + _
                (Clientes(i, 2) - Locais(j, 2)) ^ 2)
        ' Cria a variável "y_i_j" com custo "dist(i,j)"
        variavel = "y_" & i & "_" & j
        erro = UFFLP_AddVariable_(problema, variavel, 0, 1, dist, UFFLP_Binary)
    Next
Next

' Variáveis "x_j": binárias onde 1 indica o local j está aberto
For j = 1 To m
    ' Cria a variável "x_j" com custo zero
    variavel = "x_" & j
    erro = UFFLP_AddVariable_(problema, variavel, 0, 1, 0, UFFLP_Binary)
Next
```

Criação das restrições

```
' Para cada cliente i, "Soma(j=1..m) y_i_j = 1"
Dim restricao As String
For i = 1 To n
    restricao = "Cliente_" & i
    For j = 1 To m
        variavel = "y_" & i & "_" & j
        erro = UFFLP_SetCoefficient_(problema, restricao, variavel, 1)
    Next
    erro = UFFLP_AddConstraint_(problema, restricao, 1, UFFLP_Equal)
Next

' Para cada cliente i e cada local j, "y_i_j - x_j <= 0"
For i = 1 To n
    For j = 1 To m
        restricao = "Abertura_" & j & "_" & i
        variavel = "y_" & i & "_" & j
        erro = UFFLP_SetCoefficient_(problema, restricao, variavel, 1)
        variavel = "x_" & j
        erro = UFFLP_SetCoefficient_(problema, restricao, variavel, -1)
        erro = UFFLP_AddConstraint_(problema, restricao, 0, UFFLP_Less)
    Next
Next

' "Soma(j=1..m) x_j = p"
restricao = "pMedianas"
For j = 1 To m
    variavel = "x_" & j
    erro = UFFLP_SetCoefficient_(problema, restricao, variavel, 1)
Next
erro = UFFLP_AddConstraint_(problema, restricao, p, UFFLP_Equal)
```

Roteiro – Aula 2

- Modelos avançados com UFFLP
- Separação de cortes / branch-and-cut
  - Exemplo 3: Problema da Soma Máxima
  - Exemplo 4: Problema do Caixeiro Viajante
- Geração de colunas
  - Exemplo 5: Problema do Bin Packing

Exemplo 3 – Problema da Soma Máxima

- Instância fixa de exemplo:
- 8 números:

  13332 223442 83435 374351

  252342 75312 282632 263721

- Encontrar um subconjunto cuja soma seja máxima sem ultrapassar

  1143641

A solução ótima soma 1132767

Adicionando Cortes no UFFLP

1. UFFLP_SetCutCallBack
2. UFFLP_GetSolution *†
3. UFFLP_SetCoefficient *
4. UFFLP_AddConstraint *
5. UFFLP_PrintToLog *

* Dentro de uma função de “call back” de geração de cortes

† Retorna a solução da relaxação linear do nó corrente da árvore de B&B

Adicionando Cortes no UFFLP

```
' Cria o problema
problema = UFFLP_CreateProblem(UFFLP_Maximize)

' Cria variáveis da formulação
result = UFFLP_AddVariable_(problema, "x1", 0, 1, 13332, UFFLP_Binary)
result = UFFLP_AddVariable_(problema, "x2", 0, 1, 223442, UFFLP_Binary)
result = UFFLP_AddVariable_(problema, "x3", 0, 1, 83435, UFFLP_Binary)
result = UFFLP_AddVariable_(problema, "x4", 0, 1, 374351, UFFLP_Binary)
result = UFFLP_AddVariable_(problema, "x5", 0, 1, 252342, UFFLP_Binary)
result = UFFLP_AddVariable_(problema, "x6", 0, 1, 75312, UFFLP_Binary)
result = UFFLP_AddVariable_(problema, "x7", 0, 1, 282632, UFFLP_Binary)
result = UFFLP_AddVariable_(problema, "x8", 0, 1, 263721, UFFLP_Binary)

' Cria a restrição de soma máxima
result = UFFLP_SetCoefficient_(problema, "maxsum", "x1", 13332)
result = UFFLP_SetCoefficient_(problema, "maxsum", "x2", 223442)
result = UFFLP_SetCoefficient_(problema, "maxsum", "x3", 83435)
result = UFFLP_SetCoefficient_(problema, "maxsum", "x4", 374351)
result = UFFLP_SetCoefficient_(problema, "maxsum", "x5", 252342)
result = UFFLP_SetCoefficient_(problema, "maxsum", "x6", 75312)
result = UFFLP_SetCoefficient_(problema, "maxsum", "x7", 282632)
result = UFFLP_SetCoefficient_(problema, "maxsum", "x8", 263721)
result = UFFLP_AddConstraint_(problema, "maxsum", 1143641, UFFLP_Less)

' Configura o arquivo de log
result = UFFLP_SetLogInfo_(problema, ThisWorkbook.Path & "/" & "soma.log", 2)

' Escreve o modelo num arquivo LP
result = UFFLP_WriteLP_(problema, ThisWorkbook.Path & "/" & "soma.lp")

' Registra a subrotina de separação de cortes
result = UFFLP_SetCutCallBack(problema, AddressOf Separador)

' Resolve o problema
status = UFFLP_Solve(problema)
```

Função de “call back” de separação

```
Public Sub Separador(ByVal problema As Long)
    Dim ladoEsquerdo As Double, valor As Double, result As Long

    ' Calcula o lado esquerdo do corte para a solução da relaxação no nó atual
    result = UFFLP_GetSolution_(problema, "x1", valor)
    ladoEsquerdo = ladoEsquerdo + 4 * valor
    result = UFFLP_GetSolution_(problema, "x2", valor)
    ladoEsquerdo = ladoEsquerdo + 74 * valor
    result = UFFLP_GetSolution_(problema, "x3", valor)
    ladoEsquerdo = ladoEsquerdo + 27 * valor
    result = UFFLP_GetSolution_(problema, "x4", valor)
    ladoEsquerdo = ladoEsquerdo + 124 * valor
    result = UFFLP_GetSolution_(problema, "x5", valor)
    ladoEsquerdo = ladoEsquerdo + 84 * valor
    result = UFFLP_GetSolution_(problema, "x6", valor)
    ladoEsquerdo = ladoEsquerdo + 25 * valor
    result = UFFLP_GetSolution_(problema, "x7", valor)
    ladoEsquerdo = ladoEsquerdo + 94 * valor
    result = UFFLP_GetSolution_(problema, "x8", valor)
    ladoEsquerdo = ladoEsquerdo + 87 * valor

    ' Testa se o corte está violado
    If ladoEsquerdo > 377.1 Then
        ' Mostra mensagem no log
        result = UFFLP_PrintToLog_(problema, "Inserindo corte com violação " & (ladoEsquerdo - 377))

        ' Insere o corte
        result = UFFLP_SetCoefficient_(problema, "corte", "x1", 4)
        result = UFFLP_SetCoefficient_(problema, "corte", "x2", 74)
        result = UFFLP_SetCoefficient_(problema, "corte", "x3", 27)
        result = UFFLP_SetCoefficient_(problema, "corte", "x4", 124)
        result = UFFLP_SetCoefficient_(problema, "corte", "x5", 84)
        result = UFFLP_SetCoefficient_(problema, "corte", "x6", 25)
        result = UFFLP_SetCoefficient_(problema, "corte", "x7", 94)
        result = UFFLP_SetCoefficient_(problema, "corte", "x8", 87)
        result = UFFLP_AddConstraint_(problema, "corte", 377, UFFLP_Less)
    End If
End Sub
```

Exemplo 4 – Problema do Caixeiro Viajante

- *n* cidades p/ visitar
- Distâncias d_{ij} entre as cidades *i* e *j*
- Escolher um circuito de comprimento mínimo visitando cada cidade uma única vez e voltando ao ponto de partida.

Exemplo 4 – Problema do Caixeiro Viajante

- $G = (V,E)$ é um grafo completo onde cada vértice é uma cidade.

Variáveis:

- x_e ($e = (i,j) \in E$) = 1 se o caminho entre as cidades i e j é usado em qualquer sentido

Exemplo 4 – Problema do Caixeiro Viajante

$$\text{Min} \quad \sum_{e \in E} d_e \, x_e$$

$$\text{S.a} \quad \sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V$$

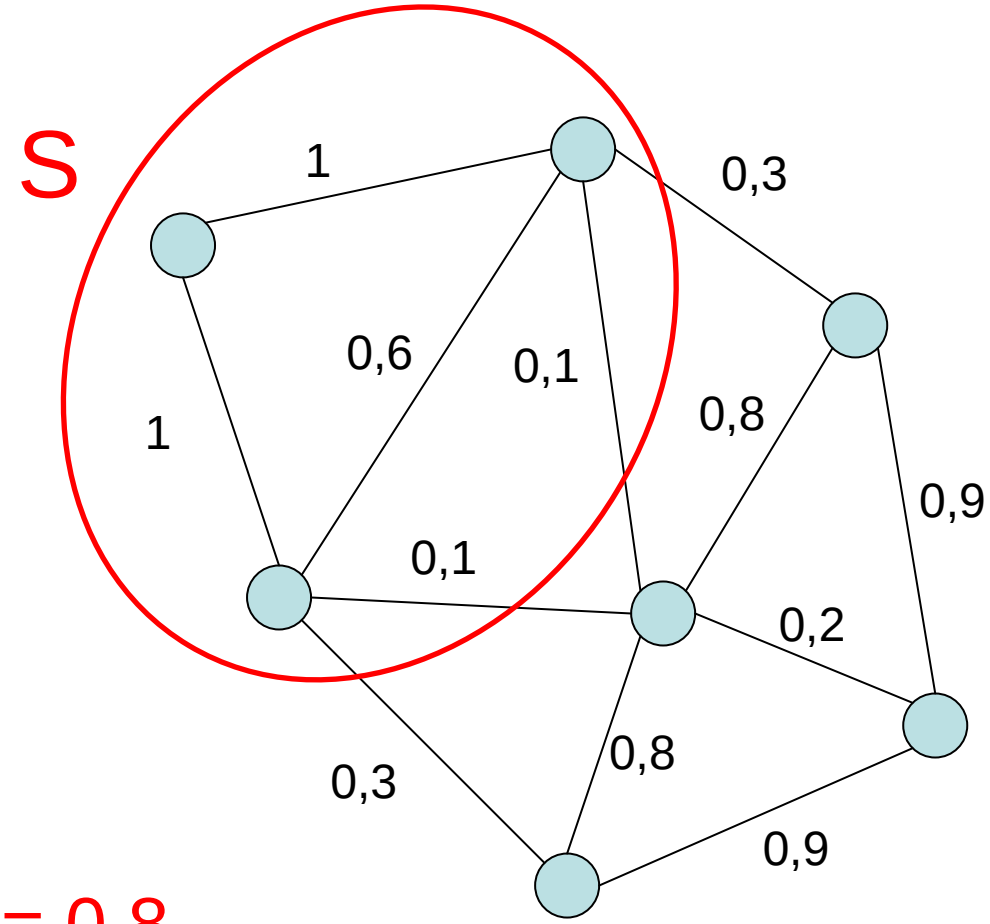
$$x_e \in \{0,1\} \qquad \forall e \in E$$

Número exponencial de restrições

Problema de Separação

- $\overline{x}_e$ ($e = (i,j) \in E$) = valor fracionário da variável x_e na solução ótima da relaxação linear.
- Encontrar $S \subset V$ que minimiza $\sum_{e \in \delta(S)} \overline{x}_e$
- Se o valor ótimo for menor que 2, o corte está violado

Problema de Separação



Valor = 0,8

Problema de Separação

Variáveis:

- w_e ($e \in E$) = 1 se e $\in$ δ(S)
- y_i (i = 1, ..., n) = 1 se i $\in$ S

Problema de Separação

$$\text{Min} \quad \sum_{e \in E} \overline{x}_e \, w_e$$

$$\text{S. a} \quad w_e \geq y_i - y_j \qquad \forall e = (i, j) \in E$$

$$w_e \geq y_j - y_i \qquad \forall e = (i, j) \in E$$

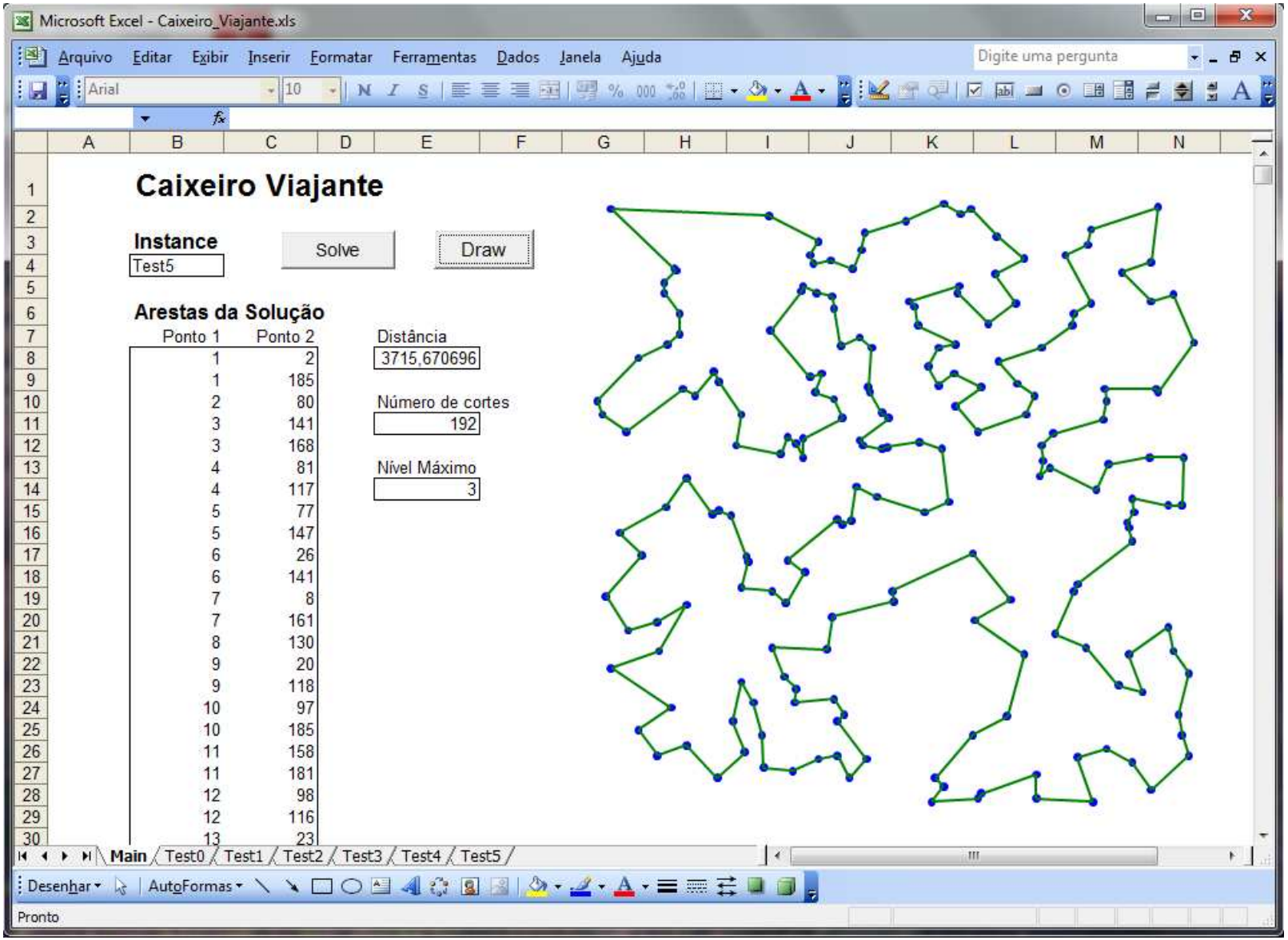
$$\sum_{i=2}^{n} y_i \leq n - 2$$

$$y_1 = 1$$

$$w_e \in \{0,1\} \qquad \forall e \in E$$

$$y_i \in \{0,1\} \qquad i = 2, \ldots, n$$

Tela Principal Excel



Criação do MIP Incompleto

```
' Create an empty problem instance
Dim problem As Long
problem = UFFLP_CreateProblem(UFFLP_Minimize)

' Create one binary variable for each possible edge
Dim varName As String
Dim error As Long
For i = 1 To n
    For j = i + 1 To n
        varName = "x_" & i & "_" & j
        error = UFFLP_AddVariable_(problem, varName, 0, 1, Distance(i, j), UFFLP_Binary)
    Next
Next

' Create one constraint for each vertex
Dim consName As String
For i = 1 To n
    ' Assembly the constraint name "point_i"
    consName = "point_" & i

    ' Set the coefficients for the constraint
    ' "SUM(j=1..i-1) x_j_i + SUM(j=i+1..n) x_i_j = 2"
    For j = 1 To n
        ' Add the variable "x_i_j" or "x_j_i" to the constraint
        If i < j Then
            varName = "x_" & i & "_" & j
        Else
            varName = "x_" & j & "_" & i
        End If
        error = UFFLP_SetCoefficient_(problem, consName, varName, 1)
    Next

    ' create the constraint
    error = UFFLP_AddConstraint_(problem, consName, 2, UFFLP_Equal)
Next
```

Criação das variáveis (Separação)

```
' Create an empty subprob instance
Dim subprob As Long
subprob = UFFLP_CreateProblem(UFFLP_Minimize)

' Create one variable "w_i_j" for each non-zero "x_i_j"
For k = 1 To UBound(sol, 1)
    varName = "w_" & sol(k).i & "_" & sol(k).j
    error = UFFLP_AddVariable_(subprob, varName, 0, 1, sol(k).x, UFFLP_Continuous)
Next

' Create one variable for each vertex
For i = 1 To n
    ' Prepare to fix only the variable "y_1" to 1
    value = 0
    If i = 1 Then value = 1

    ' Create the variable "y_i"
    varName = "y_" & i
    error = UFFLP_AddVariable_(subprob, varName, value, 1, 0, UFFLP_Binary)
Next

' Avoid all vertices inside the cut
Dim consName As String
consName = "notall"
For i = 2 To n
    varName = "y_" & i
    error = UFFLP_SetCoefficient_(subprob, consName, varName, 1)
Next
error = UFFLP_AddConstraint_(subprob, consName, n - 2, UFFLP_Less)
```

Criação das restrições (Separação)

```
' Create two constraints for each non-zero edge
Dim consName As String
For k = 1 To UBound(sol, 1)
    ' Assembly the constraint name "ext1_i_j"
    consName = "ext1_" & sol(k).i & "_" & sol(k).j

    ' Set the coefficients for the constraint "w_i_j - y_i + y_j >= 0"
    varName = "w_" & sol(k).i & "_" & sol(k).j
    error = UFFLP_SetCoefficient_(subprob, consName, varName, 1)
    varName = "y_" & sol(k).i
    error = UFFLP_SetCoefficient_(subprob, consName, varName, -1)
    varName = "y_" & sol(k).j
    error = UFFLP_SetCoefficient_(subprob, consName, varName, 1)

    ' create the constraint
    error = UFFLP_AddConstraint_(subprob, consName, 0, UFFLP_Greater)

    ' Assembly the constraint name "ext2_i_j"
    consName = "ext2_" & sol(k).i & "_" & sol(k).j

    ' Set the coefficients for the constraint "w_i_j + y_i - y_j >= 0"
    varName = "w_" & sol(k).i & "_" & sol(k).j
    error = UFFLP_SetCoefficient_(subprob, consName, varName, 1)
    varName = "y_" & sol(k).i
    error = UFFLP_SetCoefficient_(subprob, consName, varName, 1)
    varName = "y_" & sol(k).j
    error = UFFLP_SetCoefficient_(subprob, consName, varName, -1)

    ' create the constraint
    error = UFFLP_AddConstraint_(subprob, consName, 0, UFFLP_Greater)
Next
```

Inserção do corte (Separação)

```
' Check if an optimal solution has been found
If status = UFFLP_Optimal Then
    ' Get the value of the objective function
    error = UFFLP_GetObjValue_(subprob, value)

    ' Print a message in the log file
    Message = "Separation " & totalSeps & ": cut violation = " & (2 - value)
    error = UFFLP_PrintToLog_(problem, Message)

    ' Check if the cut is violated
    If value < 1.99 Then
        ' Assembly the cut name
        cutName = "cut_" & totalCuts

        ' Add all used edges to the cut
        For i = 1 To n
            For j = i + 1 To n
                ' Get the value of the variable "y_i"
                varName = "y_" & i
                error = UFFLP_GetSolution_(subprob, varName, value)

                ' Get the value of the variable "y_j"
                varName = "y_" & j
                error = UFFLP_GetSolution_(subprob, varName, value2)

                ' If the edge is in the cut, then add it
                If Abs(value - value2) > 0.5 Then
                    varName = "x_" & i & "_" & j
                    error = UFFLP_SetCoefficient_(problem, cutName, varName, 1)
                End If
            Next
        Next

        ' Add the cut to the MIP
        error = UFFLP_AddConstraint_(problem, cutName, 2, UFFLP_Greater)
```

Exemplo 5 – Problema do Bin Packing

- Dado um conjunto de n items, cada um com um peso w_i, colocá-los no menor número possível de caixas com capacidade C.

- Exemplo: n=5, w_1= 3, w_2 = 4, w_3 = 6, w_4 = 8, w_5=9 e C=10.

Exemplo 5 – Problema do Bin Packing

- Dado um conjunto de n items, cada um com um peso w_i, colocá-los no menor número possível de caixas com capacidade C.

- Exemplo: n=5, w_1= 3, w_2 = 4, w_3 = 6, w_4 = 8, w_5=9 e C=10.

São necessárias 4 caixas.

Formulação de Kantorovitch

- Seja *U* um limite superior ao número de caixas necessárias.
- Variáveis y_j indicando se a caixa j vai ser usada.
- Variáveis x_{ij} indicando que o item i vai para a caixa j.

$$\text{Min} \quad \sum_{j=1}^{U} y_j$$

$$\begin{aligned} \text{S.a} \quad \sum_{j=1}^{U} x_{ij} &= 1 && \forall i = 1 \ldots n \\ \sum_{i=1}^{n} w_i x_{ij} &\leq C.y_j && \forall j = 1 \ldots U \\ x, y &\in \{0,1\}^{n(U+1)} \end{aligned}$$

Formulação de Kantorovitch

- Essa formulação não funciona na prática
  - O limite inferior da sua relaxação linear é ruim, igual ao limite trivial $\sum_{i=1}^{n} w_i \Big/ C$.
  - No exemplo, esse limite seria 2,9.
  - A simetria das variáveis faz com que algoritmos para PI, como o branch-and-bound, sejam ineficientes.

Formulação de Gilmore-Gomory

- Defina uma variável para cada uma das *Q* possíveis combinações de itens em caixas. O número *Q* pode ser muito grande!

- Com n=5, w_1= 3, w_2 = 4, w_3 = 5, w_4 = 8, w_5=9 e C=10; são 8 combinações: {1}, {2}, {3}, {4}, {5}, {1,2}, {1,3} {2,3}.

Formulação de Gilmore-Gomory

- Defina o coeficiente a_{ij} como sendo 1 se o item *i* está na combinação *j* e 0 caso contrário.

$$\text{Min} \quad \sum_{j=1}^{Q} \lambda_j$$

$$\text{S.a} \quad \sum_{j=1}^{Q} a_{ij}\lambda_j = 1 \qquad \forall i = 1 \ldots n$$

$$\lambda \in \{0,1\}^Q$$

Formulação de Gilmore-Gomory

- No exemplo:

$$
\begin{array}{lcccccccccc}
\text{Min} & \lambda_1 & +\lambda_2 & +\lambda_3 & +\lambda_4 & +\lambda_5 & +\lambda_6 & +\lambda_7 & +\lambda_8 & & \\
\text{S.a} & \lambda_1 & & & & & +\lambda_6 & +\lambda_7 & & = & 1 \\
 & & \lambda_2 & & & & +\lambda_6 & & +\lambda_8 & = & 1 \\
 & & & \lambda_3 & & & & +\lambda_7 & +\lambda_8 & = & 1 \\
 & & & & \lambda_4 & & & & & = & 1 \\
 & & & & & \lambda_5 & & & & = & 1 \\
 & & & & & & & & \lambda & \in & \{0,1\}^8
\end{array}
$$

Formulação de Gilmore-Gomory

- No exemplo acima, o limite obtido pela relaxação linear da formulação é 3,5.

- Em geral, os limites dessa relaxação são extremamente fortes.
  - Raramente se encontra uma instância em que o limite arredondado para cima não iguale o valor da solução ótima.
  - Nunca se achou uma instância em que o limite arredondado para cima estivesse a mais de 1 unidade da solução ótima!

Resolvendo a relaxação linear por Geração de Colunas

Seja R um (pequeno) subconjunto das variáveis λ suficiente para que o seguinte PL mestre tenha solução:

$$\begin{aligned} \text{Min} \quad & \sum_{j \in R} \lambda_j \\ \text{S.a} \quad & \sum_{j \in R} a_{ij} \lambda_j = 1 \quad \forall i = 1 \ldots n \\ & \lambda \geq 0 \end{aligned}$$

Seja π o vetor de variáveis duais ótimas

Subproblema de Pricing

$$\begin{aligned} \text{Min} \quad & 1 - \sum_{i=1}^{n} \pi_i x_i \\ \text{S.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq C \\ & x \in \{0,1\}^n \end{aligned}$$

Esse é um clássico problema da mochila, que é NP-difícil, mas muito bem resolvido na prática.

Enquanto o valor da solução do subproblema for negativo, a variável correspondente é adicionada ao conjunto R e o PL mestre é resolvido novamente. Caso contrário, a solução do PL mestre é a solução da relaxação da formulação G-G.

Obtendo boas soluções inteiras

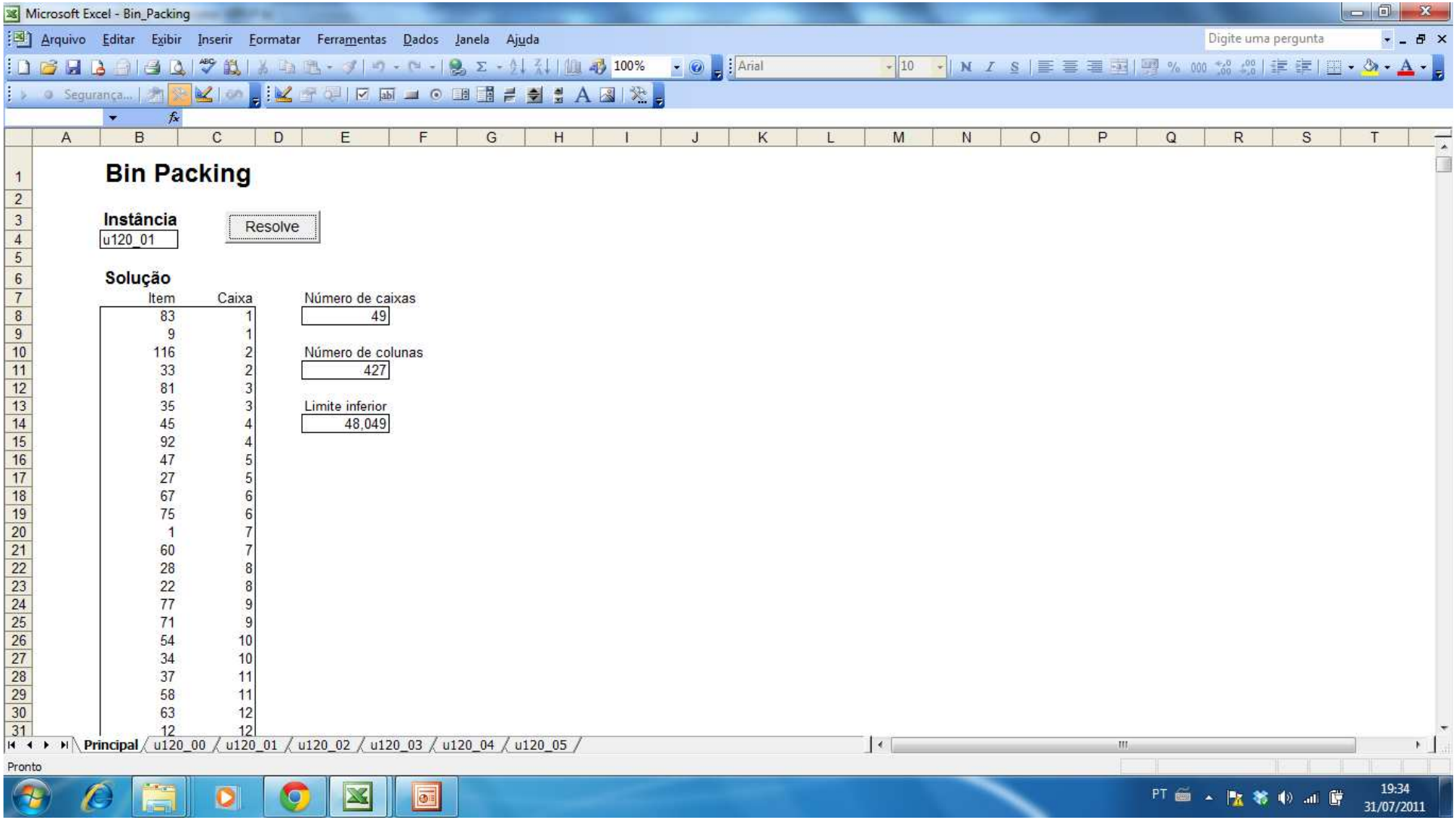
Uma possível maneira de encontrar boas soluções inteiras para o problema do bin packing é resolver um MIP apenas com as variáveis do conjunto *R* final.

O limite inferior encontrado pela relaxação muitas vezes é suficiente para provar que essa solução é ótima.

Outras Funções UFFLP usadas no exemplo

- UFFLP_GetDualSolution
- UFFLP_ChangeVariableType

Tela principal Excel



Criação do PL Mestre

```
'========= CRIAÇÃO DO PL MESTRE INICIAL - USANDO UMA SOLUÇÃO HEURÍSTICA ============

' Cria uma instância UFFLP do problema
mestre = UFFLP_CreateProblem(UFFLP_Minimize)

For j = 1 To numColunas
    variable = "lambda_" & j
    error = UFFLP_AddVariable_(mestre, variable, 0, UFFLP_Infinity, 1, UFFLP_Continuous)
Next

' Coloca os coeficientes de todos os itens das caixas criadas
For j = 1 To numColunas
    For k = 1 To caixa(j).numItens
        i = caixa(j).item(k)
        ' Coloca o coeficiente do item i na caixa j
        constraint = "Item_" & i
        variable = "lambda_" & j
        error = UFFLP_SetCoefficient_(mestre, constraint, variable, 1)
    Next
Next

' Acrescenta todas as restrições já que todos os coeficientes já estão colocados
For i = 1 To n
    constraint = "Item_" & i
    error = UFFLP_AddConstraint_(mestre, constraint, 1, UFFLP_Equal)
Next
```

Criação do Subproblema

```
'==================== CRIAÇÃO DO SUBPROBLEMA DA MOCHILA ======================

' Cria uma instância UFFLP do problema
subprob = UFFLP_CreateProblem(UFFLP_Minimize)

' Cria uma variável "x_i" para cada item i
For i = 1 To n
    ' Pega a variável dual associada ao item i no mestre restrito
    constraint = "Item_" & i
    error = UFFLP_GetDualSolution_(mestre, constraint, value)

    ' Cria a variável "x_i" no subproblema
    variable = "x_" & i
    error = UFFLP_AddVariable_(subprob, variable, 0, 1, -value, UFFLP_Binary)
Next

' Cria a restrição da capacidade da caixa "Soma(i=1..n) x_i = 1"
constraint = "Capacidade"
For i = 1 To n
    variable = "x_" & i
    error = UFFLP_SetCoefficient_(subprob, constraint, variable, peso(i))
Next
error = UFFLP_AddConstraint_(subprob, constraint, capacidade, UFFLP_Less)
```

Adição de Variável (coluna)

```
' Acrescenta mais uma coluna ao mestre restrito
numColunas = numColunas + 1
For i = 1 To n
    ' Testa se o item i está na caixa
    variable = "x_" & i
    error = UFFLP_GetSolution_(subprob, variable, value)
    If value > 0.99 Then
        ' Guarda mais um item na caixa
        caixa(numColunas).numItens = caixa(numColunas).numItens + 1
        caixa(numColunas).item(caixa(numColunas).numItens) = i
        ' Acrescenta o coeficiente da coluna no problema mestre
        variable = "lambda_" & numColunas
        constraint = "Item_" & i
        error = UFFLP_SetCoefficient_(mestre, constraint, variable, 1)
    End If
Next
variable = "lambda_" & numColunas
error = UFFLP_AddVariable_(mestre, variable, 0, UFFLP_Infinity, 1, UFFLP_Continuous)
```

Notar a simetria entre a adição de restrições e de variáveis

Resolve o Mestre Restrito Final como um MIP

```
' Altera os tipos das variáveis do mestre restrito para binárias
For j = 1 To numColunas
    variable = "lambda_" & j
    error = UFFLP_ChangeVariableType_(mestre, variable, UFFLP_Binary)
Next j

' Resolve o MIP restrito como heurística
status = UFFLP_Solve(mestre)
```

Obrigado!