

Unicamp – Universidade Estadual de Campinas
IC – Instituto de Computação
MC 548 – Projeto e Análise de Algoritmos II (turma B)
Projeto Final – Programação Linear Inteira
Prof. Zanoni Dias

Ana Carolina Merighe – RA 008018
Fábio Scramim Rigo – RA 008632

PROBLEMA 1 – Equicorte mínimo

1. Descrição das variáveis do modelo

Esse problema utiliza as seguintes variáveis:

Do arquivo de entrada (lido através da variável nomearq)

n: o número de vértices do grafo de entrada

arestas: as arestas do grafo de entrada

As variáveis definidas para resolução do problema:

Uest: é um vetor de *mpvars* (variáveis de decisão) com *n* elementos. Contém valores binários que indicam se o vértice da posição *i*, $1 \leq i \leq n$, está presente no conjunto U^* encontrado

deltaUest: é uma matriz de *mpvars* de dimensão $n \times n$. Contém valores binários que indicam se uma aresta está contida no corte definido pelo conjunto U^* encontrado. Para que um valor dessa matriz seja igual a 1, a aresta deve estar presente no grafo original, ou seja, estar descrita no arquivo de entrada e foi lido na variável *arestas*

i_E_j_em_Uest: é uma matriz de *mpvars* de dimensão $n \times n$, cuja função é auxiliar na determinação das arestas do corte. Contém valores binários que indicam se os dois vértices *i* e *j* (que correspondem aos índices da matriz) estão presentes no conjunto U^* . Sua utilização será melhor explicada na parte de restrições do modelo.

ModDeltaUest: é uma variável *linctr* (variável de cálculo sobre variáveis de decisão) que armazena a quantidade de arestas contidas no corte. Trata-se do valor que o problema deve minimizar.

2. Descrição das restrições

O problema foi modelado com apenas 3 conjuntos de restrições, descritos a seguir.

Todas as variáveis de decisão são binárias: apesar de intuitivamente precisarmos definir apenas qual é o conjunto U^* para obtermos todos os demais dados, uma forma simples de trabalhar com o software **XPress** é definir mais variáveis de decisão, restringindo seus valores para chegar ao resultado esperado. Dessa forma, o conjunto de instruções abaixo garante que todas as variáveis *mpvar* definidas podem assumir apenas os valores 0 ou 1.

```

forall (i in 1..n) Uest(i) is_binary
forall (i in 1..n, j in 1..n) do
    deltaUest(i,j) is_binary
    i_E_j_em_Uest(i,j) is_binary
end-do

```

|U*| deve ser igual ao piso de $n/2$: Essa restrição define o próprio conceito de equicorte. Deve-se encontrar um conjunto de vértices (que seja aproximadamente metade do total) para integrar o conjunto U*, e os vértices restantes devem permanecer excluídos do mesmo. A instrução abaixo garante essa restrição, já que somando os elementos "1" do vetor de vértices presentes em U* consegue-se contar quantos elementos fazem parte desse conjunto.

```

sum(i in 1..n) (Uest(i)) = floor(n/2)

```

Uma aresta está no corte se e somente se um vértice está em U* e outro não: Essa é a restrição mais difícil de implementar, uma vez que a forma mais intuitiva de fazê-la é utilizar um bloco *if-then-else*, que infelizmente não funciona quando variáveis de decisão são utilizadas como parte da validação. Implementou-se essa restrição através da definição dos valores das variáveis de decisão deltaUest e i_E_j_em_Uest. Isso é feito no bloco de instruções abaixo, que será explicado passo a passo:

```

forall (i in 1..n, j in 1..n | arestas(i,j) = 1) do
    i_E_j_em_Uest(i,j) <= 0.5*Uest(i) + 0.5*Uest(j)
    deltaUest(i,j) >= 0.1*Uest(i) + 0.1*Uest(j) - 0.2*i_E_j_em_Uest(i,j)
end-do

```

O laço *forall* é um laço de repetição em 2 variáveis – i e j – que somente executa o código interno caso o valor de arestas(i,j) for igual a "1". Portanto somente serão restringidos os valores das células da matriz que correspondem a posições onde efetivamente existe uma aresta no grafo de entrada. Os demais valores assumirão o valor "0" em deltaUest e o valor "1" em i_E_j_em_Uest, já que no momento da execução do simplex a tendência do *mosel* será minimizar a somatória dos elementos de deltaUest.

Do parágrafo anterior cabe uma explicação sobre uma característica da programação linear em linguagem *mosel* muito explorada para a restrição de valores em variáveis de decisão. No ato de definir as restrições que relacionam duas variáveis do tipo *mpvar* sempre levou-se em conta qual é o valor para o qual as variáveis se dirigem caso nenhuma restrição for aplicada a elas. No caso de deltaUest o valor natural é "0", pois a minimização de modDeltaUest faz com que o software sempre prefira atribuir "0" sempre que não for forçado a fazer o contrário. Pensamento semelhante se aplica a i_E_j_em_Uest.

Analisando por fim as restrições internas ao laço *forall* individualmente, nota-se o seguinte:

- i_E_j_em_Uest(i,j) assumirá o valor "0" se um ou mais vértices i e j NÃO pertencerem a U*. Isso ocorre devido a restrição " \leq ", que apenas permite o valor "1" caso ambos os vértices estejam em U*. Nesse caso a variável fica livre (o software pode escolher tanto "0" quanto "1") e pela razão explicitada no parágrafo anterior, escolherá "1"
- deltaUest(i,j) assumirá o valor "1" se apenas um vértice dentre i e j estiverem em U*. Isso ocorre pois se nenhum pertencer ao conjunto, todas as parcelas da segunda equação valerão 0, e se ambas pertencerem, haverá uma soma $0,1 + 0,1 - 0,2$ que resulta em 0. Isso deixa a variável livre (devido a restrição " \geq ") e o software escolherá "0". Por outro lado, se apenas uma variável estiver em U*, não haverá a subtração, que somente ocorre caso ambos os vértices estejam em U*. O valor da soma será 0,1 e a restrição " \geq " força o valor dessa célula a ser "1".

3. Descrição da função objetivo

A função objetivo é minimizar o módulo de deltaUest, armazenado em modDeltaUest, através da expressão

$$\text{modDeltaUest} := \sum(i \text{ in } 1..n, j \text{ in } 1..n) (\text{deltaUest}(i,j))$$

Essa somatória calcula o módulo do conjunto de arestas do corte, deltaUest. Cada aresta presente no corte contribui com uma unidade.

PROBLEMA 2 – Coleta de lixo

1. Descrição das variáveis do modelo

Esse problema utiliza as seguintes variáveis:

Do arquivo de entrada (lido através da variável nomearq)

m: locais potenciais para instalação de depósitos de lixo

n: setores a serem servidos pelos depósitos

custo_km: custo de transportar um metro cúbico de lixo por um quilômetro

custo_fixo: o custo fixo de instalação de um depósito, caso se concretize

capacidade: o quanto cada local potencial suporta receber de lixo anualmente, em milhares de m3

req: a demanda anual de cada setor por coleta de lixo, em milhares de m3

distancia: distância em kms entre o centro geográfico de um setor e um depósito

As variáveis definidas para resolução do problema:

setor_i_local_j: é uma matriz *n x m* de *mpvars*. Contém valores binários que indicam se o setor *i* tem seu lixo coletado pelo depósito localizado em *j*

local_usado: é um vetor de *mpvars* com *m* elementos. Contém valores binários que indicam se um local foi alocado para algum setor e, por essa razão, foi usado para instalação de um depósito. Será importante na definição do custo de implantação, já que o custo fixo deverá ser considerado somente se o local foi efetivamente usado para a implantação de um depósito

custo_total: uma variável *linctr* que armazena o custo total de implantação e operação. Minimizar essa variável constitui na função objetivo

calc_uso: é um vetor de *m* elementos cuja função é apenas dar clareza para uma das restrições a serem descritas

2. Descrição das restrições

O problema foi modelado com 4 conjuntos de restrições, descritos a seguir

As variáveis de decisão são binárias: da mesma forma que no problema do equicorte mínimo, intuitivamente precisaríamos somente de uma variável de decisão (*setor_i_local_j*), mas definiu-se *local_usado* para evitar o uso de cláusulas if-then-else. Dessa forma:

$$\begin{aligned} \text{forall } (j \text{ in } 1..m) \text{ local_usado}(j) \text{ is_binary} \\ \text{forall } (i \text{ in } 1..n, j \text{ in } 1..m) \text{ setor_i_local_j}(i,j) \text{ is_binary} \end{aligned}$$

Um local é usado se existe ao menos um setor tem seu lixo levado para ele: a variável auxiliar *calc_uso* armazena a quantidade total de setores servidos por um local, para cada local considerado. A posterior multiplicação pelo literal 0,01 ocorre para garantir que o valor resultante

esteja entre 0 e 1. A definição desse valor foi empírica, considerando-se que dificilmente trabalhar-se-á com mais de 100 locais. Caso esse valor seja superado, basta reajustar o fator.

```
forall (j in 1..m) do
    calc_uso(j) := sum (i in 1..n) (setor_i_local_j(i,j))
    local_usado(j) >= 0.01*calc_uso(j)
end-do
```

A minimização do custo fará com que o valor escolhido para local_usado seja "0" caso nenhuma restrição seja imposta. Por isso, apenas se calc_uso for igual a "0" é que local_usado ficará livre e assumirá esse valor.

Um setor só pode recolher seu lixo para um depósito: Essa restrição do enunciado é garantida pelo bloco abaixo. Para cada setor, a somatória da atribuição de depósitos deve ser igual a 1, ou seja, somente um depósito pode ser atribuído ao setor em questão:

```
forall (i in 1..n) sum (j in 1..m) (setor_i_local_j(i,j)) = 1
```

A soma da demanda de todos os setores alocados a um depósito deve ser inferior à capacidade do depósito: Essa restrição do enunciado é garantida pelo bloco abaixo:

```
forall (j in 1..m)
    (sum (i in 1..n) (setor_i_local_j(i,j)*req(i))) <= capacidade(j)
```

A iteração é feita por depósito. A multiplicação de setor_i_local_j por req resulta em zero caso o setor não foi atendido pelo depósito, logo o resultado da somatória será a soma das demandas dos setores atendidos. O total da demanda atribuída ao depósito não pode superar a sua capacidade anual de armazenar o lixo, o que é garantido pela condição "<=".

3. Descrição da função objetivo

A função objetivo é minimizar o custo total de instalação e operação. Tal custo é calculado através da expressão

```
custo_total := sum (j in 1..m) (local_usado(j)*custo_fixo(j)) +
sum (i in 1..n, j in 1..m) (custo_km*distancia(i,j)*req(i)
*1000*setor_i_local_j(i,j))
```

A somatória da primeira parcela representa o custo total de instalação de um depósito. Se o local foi usado, seu custo fixo é computado. Já a somatória da segunda parcela representa o custo de operação (transporte de lixo). Para cada setor atribuído a um depósito, computa-se a distância percorrida do centro geográfico do setor até o depósito, multiplicando-a pelo custo/km de transporte por metro cúbico de lixo, multiplicado pela quantidade de lixo transportada (req) e pelo fator 1000 de adequação de unidades (req é dado em milhares de metros cúbicos, enquanto o custo é dado em metros cúbicos).

PROBLEMA 3 – Cadeias de restaurante

A) Não instalar restaurantes da mesma cadeia em locais distantes menos de 5 km

Sob essa condição, apesar de não ser permitido instalar restaurantes de mesma cadeia distantes menos de 5 km, pode-se instalar a menos dessa distância restaurantes de cadeias distintas.

Ainda é válido salientar que a restrição aqui imposta corresponde estritamente ao anunciado geral aos dois itens do exercício. Por esse motivo, no item B tem-se somente o acréscimo de mais uma condição, em acréscimo àquelas aqui definidas.

1. Descrição das variáveis do modelo

Para leitura dos dados de entrada, são utilizadas as seguintes variáveis:

nomearq: string referente ao nome do arquivo de entrada

n: inteiro que indica o número de possíveis locais para instalação dos restaurantes

lucro: array bidimensional $n \times 2$ ($lucro(i,j)$) que contém o valor do lucro mensal presumido, em reais, para a instalação de um restaurante da cadeia *j* no local *i*

Menos5: matriz bidimensional $n \times n$, na qual o valor que constar na posição $Menos5(i,j)$ será um se e somente se os locais *i* e *j* estiverem distantes menos de 5 km

Variáveis de decisão (*mpvar*):

cadFF: array de mpvars binários com dimensão *n*, que na posição *i* será 1 se e somente se for instalado um restaurante da cadeia de fast food no local *i*

cadLC: array de mpvars binário com dimensão *n*, que na posição *i* será 1 se e somente se for instalado um restaurante da cadeia de a la carte no local *i*

Variáveis auxiliares (*linctr*):

lucro_total: indica o lucro total de uma configuração de instalação das cadeias

2. Descrição das restrições do problema

Abaixo segue uma descrição das restrições adotadas para a resolução do problema.

O bloco abaixo garante que os elementos dos arrays de decisão descritos acima sejam binários:

```
forall (i in 1..n) do
    cadFF(i) is_binary
    cadLC(i) is_binary
end-do
```

Com a restrição a seguir garante-se que em um mesmo local *i* seja instalada somente uma das opções do grupo, i.e., se um local for utilizado para instalar uma unidade da rede de fast food, aí não poderá ser instalado um restaurante da rede a la carte e vice-versa:

```
forall (i in 1..n) cadFF(i) + cadLC(i) <= 1
```

Na restrição a seguir, relacionam-se três das variáveis declaradas anteriormente, de modo a garantir que não se possa instalar dois restaurantes da mesma rede a menos de 5km de distância. Note que nas duas relações abaixo, quando a distância entre os locais *i* e *j* for maior maior do que 5 km, as duas condições são sempre aceitas. Quando a distância entre *i* e *j* for inferior a 5 km ($Menos5(i,j) = 1$), para que as relações sejam satisfeitas só podemos ter um restaurante de uma das cadeias instalado em *i* ou em *j*. Se a essa distância tivéssemos dois restaurantes da mesma cadeia (um em *i* e outro em *j*), a soma iria para 2, o que feriria a restrição.

```
forall (i in 1..n, j in 1..n) do
    Menos5(i,j) * (cadFF(i) + cadFF(j)) <= 1
    Menos5(i,j) * (cadLC(i) + cadLC(j)) <= 1
end-do
```

2. Descrição da função objetivo

Por fim, temos a função objetivo a ser maximizada:

$$lucro_total := \sum(i \text{ in } 1..n) (lucro(i,1)*cadFF(i) + lucro(i,2)*cadLC(i))$$

À variável auxiliar `lucro_total` finalmente é atribuído o valor dos lucros obtidos pela configuração a ser adotada para a cadeia de fast food e para a cadeia de serviço a la carte, que corresponde a soma de todos os lucros presumidos para cada local i em que é instalado um restaurante de uma determinada cadeia.

Na variável `lucro(i,j)` temos o valor do lucro presumido para a cadeia j no local i , sendo que a coluna $j=1$ apresenta os valores para fast food e a $j=2$ os lucros para a cadeia de a la carte. Dessa maneira, se `cadFF(i) = 1` (restaurante instalado em i), produto `lucro(i,1)*cadFF(i)` fornecerá o valor do lucro esperado para a cadeia de fast food no local i . Se `cadFF(i) = 0`, o produto também será 0 e o valor esperado para i não entrará no cômputo do lucro referente a esta cadeia. A soma de todos os valores combinados dessa forma resultará no lucro total esperado para fast food.

O mesmo raciocínio se aplica à `lucro(i,2)*cadLC(i)` e à soma de seus termos.

A soma desses valores representa o lucro total esperado para uma dada distribuição de restaurantes do grupo e é esse valor que o grupo controlador das cadeias deseja maximizar (`maximize(XPRS_PRI, lucro_total)`).

B) Ao se instalar um restaurante de uma rede em um local, obrigatoriamente instalar uma restaurante da outra a menos de 5km de distância, mantida a estratégia definida em A.

Pelas razões descritas acima, essa solução apresenta exatamente as mesmas declarações de variáveis e função objetivo do item A, o que dispensa novos esclarecimentos acerca desses tópicos.

Analogamente, as restrições descritas em A permanecem inalteradas. No entanto, foi necessária a inclusão de uma nova para garantir a condição solicitada neste enunciado.

Com a restrição

```
forall (i in 1..n) do
    sum (j in 1..n) (Menos5(i,j)*cadFF(j)) >= cadLC(i)
    sum (j in 1..n) (Menos5(i,j)*cadLC(j)) >= cadFF(i)
end-do
```

garante-se que ao instalar uma filial de uma cadeia, há de se instalar também uma filial da outra a uma distância não inferior a 5km. Deve-se salientar que segundo o enunciado não há restrições quanto ao número de restaurantes da segunda cadeia nesse raio, desde que respeitadas as condições definidas anteriormente.

Dada a atribuição de um local i para uma dada cadeia, a condição força que exista ao menos aquela atribuição para a outra. Assim, se houver um restaurante de uma das cadeias em i , necessariamente deve-se ter ao menos um (mas não exatamente um) da outra, o que é garantido pelo `>=`. Ou seja, para cada local i , dentre os n possíveis, a soma de todos os restaurantes de uma determinada rede que distam menos de 5km (`Menos5(i,j) = 1`) daquele local inicial deve ser maior ou igual à presença da outra cadeia em i . Se `Menos5(i,j)=0`, a restrição é liberada pelo igual.

A condição é feita utilizando a mesma lógica para as duas situações: quando o primeiro é fast food e quando aí temos um a la carte.

PROBLEMA 4 – Caminhão tanque

1. Descrição das variáveis do modelo

Esse problema utiliza as seguintes variáveis:

Do arquivo de entrada (lido através da variável nomearq)

m: quantidade de tipos de combustíveis diferentes a serem entregues

n: quantidade de compartimentos presentes no caminhão tanque

capacidade: capacidade de armazenamento de cada compartimento

demanda: demanda total por cada tipo de combustível

As variáveis definidas para resolução do problema:

quant_comb_compart: é uma matriz $m \times n$ de *mpvars*. Contém valores inteiros que indicam a quantidade de combustível do tipo *i* é armazenada no compartimento *j*. Trata-se da variável de decisão principal

uso_comb_compart: é uma matriz $m \times n$ de *mpvars*. Contém valores binários que indicam se o compartimento foi utilizado para transporte de combustível ou se ele irá vazio

sum_quantidades: uma variável *linctr* que armazena o total de litros transportados pelo caminhão. Usada por motivo de clareza

dem_n_atendida: uma variável *linctr* que armazena o total de litros demandados, porém não entregues. Minimizar essa variável constitui na função objetivo.

2. Descrição das restrições

O problema foi modelado com 5 conjuntos de restrições, descritos a seguir

Especificação dos tipos das variáveis de decisão: Ao contrário dos anteriores, o problema presente tem um conjunto de variáveis de decisão inteiras. Por essa razão, a especificação de domínio dela é ligeiramente diferente:

```
forall (i in 1..m, j in 1..n) do
    quant_comb_compart(i,j) is_integer
    uso_comb_compart(i,j) is_binary
end-do
```

Um compartimento é marcado como utilizado por um combustível se a quantidade que carrega é maior que 0: É a restrição que garante a correção dos valores armazenados em *uso_comb_compart*. O valor de *uso_comb_compart* é "1" somente se alguma quantidade de combustível for transportada no compartimento. Isto é:

```
forall (i in 1..m, j in 1..n) do
    quant_comb_compart(i,j) <= uso_comb_compart(i,j) * 1000000
end-do
```

O número 1000000 é um número arbitrariamente grande que garante que o lado direito da desigualdade sempre vai ser maior que o lado esquerdo, caso *uso_comb_compart* seja igual a "1". A única forma de a variável *uso_comb_compart* ser "0" é fazer com que *quant_comb_compart* seja também "0".

Um compartimento pode ser usado por no máximo um combustível: Para toda coluna de *uso_comb_compart*, no máximo uma célula pode estar marcada como usada. Isso garante que ou o compartimento não carrega combustível algum, ou carrega apenas um combustível, conforme bloco abaixo:

```
forall (j in 1..n) do
    sum (i in 1..m) (uso_comb_compart(i,j)) <= 1
end-do
```

A somatória das quantidades de um tipo de combustível transportado deve ser no máximo igual a sua demanda: O caminhão não pode transportar mais combustível do que o demandado. Logo, a soma de todas as quantidades de um único tipo de combustível transportado não deve superar a demanda para esse combustível em particular.

```
forall (i in 1..m) do
    sum (j in 1..n) (quant_comb_compart(i,j)) <= demanda(i)
end-do
```

Limita a capacidade de cada compartimento: um compartimento tampouco pode carregar mais combustível do que comporta. Devido à restrição de um combustível por compartimento, apenas uma célula da coluna terá valor diferente de zero. Logo a razão da somatória é apenas a garantia de que todos os combustíveis foram considerados (e verificados) nessa restrição.

```
forall (j in 1..n) do
    sum (i in 1..m) (quant_comb_compart(i,j)) <= capacidade(j)
end-do
```

3. Descrição da função objetivo

A função objetivo é minimizar a quantidade de combustível não entregue (dem_n_atendida). Tal quantidade é calculada através das expressões:

```
sum_quantidades := sum (i in 1..m, j in 1..n) (quant_comb_compart(i,j))
dem_n_atendida := (sum(i in 1..m) (demanda(i))) - sum_quantidades
```

Na primeira equação, soma-se todo o combustível transportado. Na segunda, subtrai-se esse total da demanda total calculada pela somatória da primeira parcela.