

Trabalho Prático MC548

Projeto e Análise de algoritmos II

Prof. Zanoni

Eduardo Madeira Fleury RA015914
Marcelo Alejandro Aoki Fraile RA016739

Neste trabalho encontramos quatro problemas para serem resolvidos em PLI(Programação Linear inteira) utilizando o software Xpress. Seguem as descrições das variáveis, as restrições e função objetivo dos 4 problemas:

Questão 1

Variáveis do modelo

particao : array(1..n) of mpvar (binárias)

Representa a divisão dos N vértices do grafo original em dois conjuntos, aqueles em que "particao(i) = 1" são os vértices do conjunto U^* lembrando que $|U^*|$ deve ser igual à "floor(n/2)".

arest_part : array(1..n,1..n) of mpvar (binárias)

Indica quais são as arestas que foram o corte, ou seja, que ligam um vértice que está em U^* a um que está fora de U^* .

Como toda matriz de vértices é simétrica em relação à diagonal principal.

Descrição das Restrições

Equicorte := sum(i in 1..n) particao(i) = floor(n/2)

O conjunto U^* deve conter mais ou menos metade do número de vértices do grafo de entrada, essa soma é igual a $|U^*|$ e portanto deve ser igual a floor(n/2).

```
forall (i in 1..n, j in 1..n | arestas(i,j) = 1) do
    XOR1(i,j) := arest_part(i,j) >= particao(i) - particao(j)
    XOR2(i,j) := arest_part(i,j) >= particao(j) - particao(i)
    XOR3(i,j) := arest_part(i,j) <= 2 - particao(i) - particao(j)
    XOR4(i,j) := arest_part(i,j) <= particao(i) + particao(j)
end-do
```

As arestas de "arest_part" são aquelas em que 1 dos vértices está dentro da partição e o outro está fora, ou seja, cada posição da matriz recebe um XOR(Vertex1, Vertex2) onde VertexX diz se o respectivo vértice está ou não na partição (i.e. se "Particao(X)" é igual a 1 ou 0).

Aplicamos a restrição àquelas arestas que já existiam no grafo de entrada pois não podemos criar novas arestas. As arestas que não existiam foram tratadas com outra restrição, explicada logo adiante.

Note que não é possível implementar um XOR utilizando a função AND (*) pois a mesma não é linear, por isso utilizamos as 4 funções lineares abaixo.

Tabela Verdade:

Restrição	(0,0)	(0,1)	(1,0)	(1,1)
$XOR(v1,v2) \geq v1 - v2$	≥ 0	≥ -1	≥ 1	≥ 0
$XOR(v1,v2) \geq v2 - v1$	≥ 0	≥ 1	≥ -1	≥ 0
$XOR(v1,v2) \leq 2 - v1 - v2$	≤ 2	≤ 1	≤ 1	≤ 0
$XOR(v1,v2) \leq v1 + v2$	≤ 0	≤ 1	≤ 1	≤ 2
Resultado	= 0	= 1	= 1	= 0

```
forall (i in 1..n, j in 1..n | arestas(i,j) = 0) arest_part(i,j) = 0
```

Todas as arestas que não existiam no grafo continuam não existindo no equicorte.

Função Objetivo

```
Custo:= sum(i in 1..n-1) sum(j in i..n) arest_part(i,j)
```

O "Custo" é igual à $|U^*|$. Como cada aresta aparece duas vezes como um número "1" em *arest_part*, sendo uma vez no triângulo superior direito da matriz e outra no inferior esquerdo, basta somar quantos "1"s existem em uma dessas regiões da matriz para obter a função objetivo a ser minimizada. Escolhemos a metade superior direita.

Questão 2

Variáveis do modelo

```
constroi : array(1..m) of mpvar (variável binária)
```

Vetor com todos os locais que podem ser utilizados que serve para sabermos se devemos ou não construir um determinado depósito. Assim, para cada *constroi(i)* temos um valor 1 ou 0 indicando ou não que o depósito deve ser construído no local.

```
coleta : array(1..n,1..m) of mpvar (variável binária)
```

Esta matriz nos mostra a relação entre um setor(que precisa descartar lixo) e um depósito. Assim, para cada par (n,m) temos uma resposta 1 ou 0 indicando ou não que um determinado setor tem que descartar seu lixo no correspondente depósito.

Descrição das restrições

```
forall (i in 1..n, j in 1..m)
    DepositoConstruido(i,j) := coleta(i,j) <= constroi(j)
```

Nessa restrição somente podem ser atribuídos setores a depósitos construídos, ou seja, $coleta(i,j) = 1$ só vale se $constroi(j) = 1$. Lembrando que se $constroi(j) = 1$, $coleta(i,j)$ pode ser igual a 0. Como já dito, essa restrição só impede a que a relação $coleta(i,j) = 1$ seja feita para casos em que o depósito não foi construído.

```
forall (i in 1..n)
    UnicoDeposito(i) := sum(j in 1..m) coleta(i,j) = 1
```

Temos aqui que, para cada setor de 1 a n, verificamos se a soma de $coleta(i,j)$ para cada um dos depósitos é igual à 1, pois só podemos ter a atribuição de um depósito para cada setor.

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

Neste caso, limitamos a quantidade de lixo que pode ser destinada para cada depósito. Assim, para cada depósito somamos a quantidade de lixo enviada por cada setor através de $coleta(i,j) * req(i)$ que mostra a quantidade de lixo para um determinado depósito. Após somarmos todo o lixo atribuído à um depósito, analisamos se o total é menor ou igual a $capacidade(j)$ do depósito.

Função Objetivo

Procuramos para nosso problema, encontrar o custo mínimo para construir depósitos e atribuir setores aos mesmos contando os custos de construção e de deslocamento(transporte do lixo). Para isso, temos que calcular 2 somas:

1-) A soma de custo fixo dos depósitos construídos

```
sum(i in 1..m) constroi(i) * custo_fixo(i)
```

2-) A soma do custo do transporte

Como a custo desta parcela é dada por km e por metro cúbico de lixo temos

```
sum(i in 1..n) sum (j in 1..m)
    distancia(i,j) * coleta(i,j) * custo_km * req(i) * 1000
```

Notamos a presença de uma multiplicação por 1000 que existe devido a capacidade ser dada em milhares de metros cúbicos.

Depois, somamos essas duas parcelas e calculamos o mínimo para essa última soma já que queremos o custo mínimo.

Custo mínimo = custo fixo + custo transporte

minimize(Custo mínimo)

Questão 3 - A

Variáveis do modelo

instala : array(1..*n*,1..2) of mpvar (variável binária)

Matriz que atribui uma restaurante a um local. Se $instala(i,j) = 1$ então temos q instalar um restaurante no determinado local.

Descrição das restrições

```
forall (i in 1..n)  
    LimiteDaFisica(i) := instala(i,1) + instala(i,2) <= 1
```

Como não podemos instalar 2 restaurantes no mesmo local, verificamos se a soma de restaurantes e lanchonetes(fast-food) em um mesmo é menor ou igual a 1.

```
forall (i in 1..n-1, j in i+1..n | Menos5(i,j) = 1)  
    ProximoDemais1(i,j) := instala(i,1) + instala(j,1) <= 1
```

Para cada par de lugares próximos(limitados por 5km $Menos5(i,j) = 1$), não podemos instalar dois restaurantes da mesma cadeia(no caso, cadeia 1). Por isso, a soma de $instala(i,1) + instala(j,1)$ tem q ser menor ou igual a 1.

```
forall (i in 1..n-1, j in i+1..n | Menos5(i,j) = 1)  
    ProximoDemais2(i,j) := instala(i,2) + instala(j,2) <= 1
```

Análogo à restrição anterior, porém para a cadeia 2.

Função Objetivo

Neste exercício procuramos maximizar o lucro através da alocação de restaurantes em diferentes locais. Para isso temos que somar o lucro das lojas instaladas:

Soma dos lucros

```
Lucro := sum(i in 1..n) sum(j in 1..2) instala(i,j) * lucro(i,j)
```

Feita essa soma temos que maximizar o valor de Lucro.

maximize(Lucro)

Questão 3 - B

Variáveis do modelo

instala : array(1..*n*,1..2) of mpvar (variável binária)

Matriz que atribui uma restaurante a um local. Se $instala(i,j) = 1$ então temos q instalar um restaurante no determinado local.

Descrição das restrições

```
forall (i in 1..n)  
    LimiteDaFisica(i) := instala(i,1) + instala(i,2) <= 1
```

Como não podemos instalar 2 restaurantes de cadeias diferentes no mesmo local, verificamos se a soma de restaurantes em um mesmo é menor ou igual a 1.

```
forall (i in 1..n-1, j in i+1..n | Menos5(i,j) = 1)  
    ProximoDemais1(i,j) := instala(i,1) + instala(j,1) <= 1
```

Para cada par de lugares próximos(limitados por 5km $Menos5(i,j) = 1$), não podemos instalar dois restaurantes da mesma cadeia(no caso, cadeia 1). Por isso, a soma de $instala(i,1) + instala(j,1)$ tem q ser menor ou igual a 1.

```
forall (i in 1..n-1, j in i+1..n | Menos5(i,j) = 1)  
    ProximoDemais2(i,j) := instala(i,2) + instala(j,2) <= 1
```

Análogo à restrição anterior, porém para a cadeia 2.

```
forall (i in 1..n)  
    sum(j in 1..n) Menos5(i,j) * instala(j,2) >= 1*instala(i,1)
```

Caso existe um restaurante da cadeia1 no local então temos q ter um restaurante da cadeia2 instalado em um lugar próximo(inferior a 5km).

```
forall (i in 1..n)  
    sum(j in 1..n) Menos5(i,j) * instala(j,1) >= 1*instala(i,2)
```

Análogo ao anterior porém para o caso de um restaurante da cadeia 2.

Função Objetivo

Neste exercício procuramos maximizar o lucro através da alocação de restaurantes em diferentes locais. Para isso temos que somar o lucro das lojas instaladas:

Soma dos lucros

```
Lucro := sum(i in 1..n) sum(j in 1..2) instala(i,j) * lucro(i,j)
```

Feita essa soma temos que maximizar o valor de Lucro.

```
maximize(Lucro)
```

Questão 4

Variáveis do modelo

```
quantidade : array (1..n, 1..m) of mpvar (inteiros)
```

Este vetor serve para saber a quantidade de combustível colocada em um tanque determinado.

```
atribuicao : array(1..n, 1..m) of mpvar (binários)
```

Neste vetor temos a relação entre um tanque e um tipo de combustível. Assim, se destinamos um tipo de combustível a um tanque, atribuição(i,j) tem q ser igual a 1 para determinados i e j.

Descrição das restrições

```
forall (i in 1..n)
    UnicoDeposito(i) := sum(j in 1..m) atribuicao(i,j) <= 1
```

Serve para que cada tanque possa conter apenas um tipo de combustível. Fazemos, desse modo, uma verificação da somatória de atribuição(i,j) para cada tanque que deve ser menor ou igual a 1.

```
forall (i in 1..n, j in 1..m)
    quantidade_deposito(i,j) := quantidade(i,j) <= atribuicao(i,j)
    * capacidade(i)
```

Aqui colocamos a restrição de capacidade dos tanques já que não podemos colocar mais combustível do que o tanque suporta. Além disso, só podemos colocar determinada quantidade em um tanque se esse combustível for realmente atribuído a aquele.

```
forall (j in 1..m) sum (i in 1..n) quantidade(i,j) <= demanda(j)
```

Outra restrição é que não podemos utilizar mais combustível do que a demanda do mesmo, ou seja, para cada combustível somamos a quantidade alocada em cada tanque e, limitamos pela demanda(j) do combustível.

Função Objetivo

Neste problema buscamos a sobra mínima de combustível. Para tanto, temos que somar toda a demanda e subtrair da quantidade de combustível alocada no caminhão. Após feita a soma, como queremos encontrar o mínimo, utilizamos a chamada *minimize()*. Temos:

Soma da demanda

```
sum (j in 1..m) demanda(j)
```

Soma dos combustíveis alocados

```
sum(i in 1..n, j in 1..m) quantidade(i,j)
```

Sobra = Soma da demanda - Soma dos combustíveis alocados

Como queremos a sobra mínima

```
minimize(Sobra)
```