

Trabalho Prático de MC548

Professor **Zanoni Dias**

Aluno: **Gabriel Rossetto**

962428

Seguem as as soluções dos problemas, seguem também em anexo os arquivos .mos com os modelos em questão.

1 Problema do Equicorte

Este problema envolve a divisão de um grafo em duas partes de tamanho praticamente igual, para isso devem ser removidas algumas arestas, o objetivo é minimizar esse numero de arestas

Variáveis

Alem das entradas definidas pelo enunciado do problema temos as seguintes variaveis :

vertices = 1..n

Usado para facilitar a leitura

vercorte: array(vertices) of

Um vetor que contem os vertices que vão fazer parte do corte, tem valor 1 o indice correspondente a um vertice que está presente no corte.

arcorte: array (vertices,vertices) of mpvar

Matriz que simula as arestas do corte, todas as arestas com o valor 1, serão retiradas do grafo para gerar o equicorte.

mincorte: lincnr

Restrições

```
forall (x in vertices, y in vertices | arestas(x,y)=1 ) do
  arcorte(x,y) is_integer
  arcorte(x,y) <=1
  arcorte(x,y) >=0
end-do
```

Restringe o valor do vetor dos vertices e os tornas discretas.

```
forall (x in vertices) do
  vercorte(x) is_integer
  vercorte(x) <=1
  vercorte(x) >=0
end-do
```

Restringe o valor da matriz das arestas e as tornas discretas.

```
forall(x in vertices, y in vertices | arestas(x,y)=1) do
  arcorte(x,y) <= 2 - vercorte(x) - vercorte(y)
  arcorte(x,y) <= vercorte(y) + vercorte(x)
  arcorte(x,y) >= vercorte(x) - vercorte(y)
  arcorte(x,y) >= vercorte(y) - vercorte(x)
end-do !
```

Restrições para evitar que dois vertices compartilhem a mesma aresta... efetivamente separando o grafo.

```
sum(x in vertices) vercorte(x) = floor(n/2)
```

Garatimos que o numero de vertices em vercorte é aproximadamente metade do numero de vertices originariamente presentes no grafo (n).

Função Objetivo

```
mincorte := sum(i in vertices, j in vertices) arcorte(i,j);
```

A função objetivo é a contagem do numero de arestas removidas pelo corte, temos que minimizar isso para conseguir o menor equicorte possivel para o grafo.

Tenta-se minimizar o corte.

2 Problema da coleta de lixo

Este problema envolve a otimização da instalação de depósitos de lixo para atender a demanda de setores geradores de lixo.

Variáveis

Alem das entradas definidas pelo enunciado do problema temos as seguintes variáveis :

depositos = 1..*m* e *setores* = 1..*n*

Que são usados apenas para facilitar a leitura dos arrays e dos contadores que usariam esses ranges.

uso: array(*setores*,*depositos*) of mpvar

É uma matrix que contém 1 nas células que ligam o setor ao depósito que ele usa.

usado : array(*depositos*) of boolean

É um array que contém verdadeiro nos depósitos que foram necessários para a solução.

custo: linctr

É usado para a função objetivo.

Restrições

```
forall(x in setores,y in depositos)
do
  uso(x,y) is_integer
  uso(x,y) >= 0
  uso(x,y) <= 1
end-do
```

Delimita os valores válidos para a matriz de uso.

```
forall (x in setores) do
  sum (y in depositos) uso(x,y) = 1
end-do
```

Essa restrição faz com que cada linha (setores) só tenha uma célula marcada, ou seja, o lixo de cada setor só pode ser levado a um único depósito. E garante que o lixo tem que ser levado para algum depósito.

```
forall (y in depositos) do
  sum (x in setores) req (x) * uso (x,y) <= capacidade (y)
end-do
```

Aqui garantimos que cada depósito não ultrapassa sua capacidade máxima (somando os requerimentos de todos os setores que enviam lixo para ele)

```
forall (y in depositos) do
  if getsol(sum (x in setores) uso (x,y)) > 0
  then usado(y) := true
```

```

    else usado(y) := false
  end-if
end-do

```

Essa restrição popula o vetor *usado* marcando aqueles depósitos que fazem parte da solução com verdadeiro.

```

forall (y in depositos) do
  if not usado(y) then
    sum (x in setores) uso(x,y) = 0
  end-if
end-do

```

Garante que aqueles depositos que não estão definidos como usados não tem setores associados a eles.

Função Objetivo

```

custo_deps := sum (y in depositos | usado(y)) custo_fixo (y)
custo_transp := sum(x in setores, y in depositos) uso(x,y) * (distancia(x,y) * req(x) * custo_km * 1000)
custo := custo_deps + custo_transp

```

Para facilitar a visualização dividi a função objetivo em duas partes, o custo fixo dos depósitos (*custo_deps*) que é a soma dos custos fixos dos depósitos usados e *custo_transp* que representa o custo de transporte do lixo do setor para o deposito que lhe foi alocado.

Tenta-se minimizar o custo.

3 Problema dos Restaurantes

Este problema envolve uma empresa com duas cadeias de restaurantes que tenta maximizar o seu lucro ao se expandir em uma nova cidade. Este problema possui uma restrição extra, que é apresentada aqui mas só se aplica à variação “b”.

Variáveis

Alem das entradas definidas pelo enunciado do problema temos as seguintes variáveis

locais = 1..*n*

Usada apenas para facilitar a leitura.

uso1 : array (*locais*) of mpvar

uso2 : array (*locais*) of mpvar

Um vetor para os restaurantes do tipo1 (fast food – *uso1*) e um os restaurantes do tipo 2 (a la carte – *uso2*) onde recebem 1 quando existe um restaurante desse tipo nesse local.

obtido: *linctr*

Usado para a função objetivo.

Restrições

```
forall (x in locais) do
  uso1(x) is_binary
  uso2(x) is_binary
  uso1(x) >= 0
  uso1(x) <= 1
  uso2(x) >= 0
  uso2(x) <= 1
  uso1(x) + uso2(x) <= 1
end-do
```

Define os valores validos para os vetores e na ultima linha, garante que não existem 2 restaurantes no mesmo lugar.

```
forall (x in locais, y in locais) do
  if Menos5(x,y) then
    uso1(x) + uso1(y) <=
    uso2(x) + uso2(y) <= 1
  end-if
end-do
```

Aqui garantimos que não existem restaurants do mesmo tipo a menos de 5km um do outro, fazendo com que a soma de dois “vizinhos” nunca seja maior que 1.

```
forall (x in locais) do
  uso1(x) <= sum (y in locais) if(Menos5 (x,y),1,0) * uso2(y)
  uso2(x) <= sum (y in locais) if(Menos5 (x,y),1,0) * uso1(y)
end-do
```

Essa restrição, presente apenas na versão “b” do problema, força a existencia de um um

restaurante da outra categoria nas proximidades para que se possa fazer um restaurante nesse local. (se a soma de todas as proximidades no outro vetor for zero, esse terá que ser igual a zero).

Função Objetivo

$obtido := \sum (x \text{ in locais}) lucro(x,1) * uso1(x) + \sum (x \text{ in locais}) lucro(x,2) * uso2(x)$

A função objetivo soma os lucros dos dois tipos de restaurantes. Para conseguir o lucro de cada tipo, se soma todos aqueles que vão ser construídos daquele tipo.

Tenta-se maximizar o lucro obtido.

4 Problema da distribuição de combustível

Neste problema uma distribuidora de combustíveis deseja maximizar a eficiência de seu único caminhão, que deve transportar diferentes tipos de combustíveis em seus vários compartimentos de tamanhos variáveis.

Variáveis

Além das entradas definidas pelo enunciado do problema temos as seguintes variáveis

tanques = 1..*n* e *combustíveis* = 1..*m*

levado : array(*combustíveis*) of mpvar

Vetor que guarda a quantidade de combustível levado de cada tipo.

deixado : array(*combustíveis*) of mpvar

Vetor que guarda a quantidade de combustível deixado de cada tipo.

uso : array (*tanques*, *combustíveis*) of mpvar

Matriz que indica qual tanque está armazenando qual tipo de combustível (valor 1 quando o tanque está armazenando o combustível em questão)

total_levado: lincstr

Usado na função objetivo.

Restrições

```
forall (x in tanques, y in combustíveis) do
  uso(x,y) is_binary
  levado(y) is_integer
  deixado(y) is_integer
end-do
```

Restringe os valores possíveis em todos os vetores.

```
forall (x in tanques) do
  sum (y in combustíveis) uso(x,y) <= 1
end-do
```

Garante que não existe mais de um tipo de combustível no mesmo tanque. (varrendo a matriz uso para que não haja mais de um)

```
forall (y in combustíveis) do
  demanda(y) = levado(y) + deixado(y)
end-do
```

Divide a demanda de combustível entre o que é levado e o que é deixado.

```
forall (y in combustíveis) do
  levado(y) <= sum(x in tanques) uso(x,y) * capacidade(x)
end-do
```

Restringe a quantidade de combustível levada em cada tanque à sua capacidade máxima.

Função Objetivo

```
total_levado := sum (y in combustiveis) levado(y)  
maximize(total_levado)  
total_deixado := sum (y in combustiveis) deixado(y)
```

Nessa função objetivo foi maximizado a quantidade de combustível levado e depois subtrai-se da quantidade total de combustível para descobrirmos quanto foi deixado.

Tenta-se maximizar a quantidade de combustível levado.