Trabalho Final

Universidade Federal da Paraíba

Centro de Informática

Grupo:

Allef Brenno Gomes De Lima, 20160121380

Júlio Raphael de Oliveira Silva, 2016085924

Link para o repositório deste projeto:

> https://github.com/JRaphaelO/Projeto-Final.git

Apresentação:

> https://youtu.be/lpKkIuAQgNo

Planejamento de Produção

Problema

Uma fábrica produz *n* produtos finais a partir *m* matérias-primas. A produção de 1 unidade do produto *i* exige a_{ij} unidades da matéria-prima *j* e b_i horas da linha de produção. Existe uma disponibilidade de *B* horas da linha de produção por mês ao custo fixo de *F* reais. Cada troca de produto na linha gasta *T* horas. Lotes fechados com L_j unidades da matéria-prima *j* podem ser comprados por C_j reais cada. Cada produto pode ser produzido ou não. No entanto, caso decida-se produzir o produto *i*, deve-se atender uma demanda mínima $DMIN_i$, e uma demanda máxima $DMAX_i$. Uma unidade do produto *i* pode ser vendida por R_i reais. Determine quanto se deve produzir de cada produto nesse mês para maximizar o lucro da fábrica.

Por exemplo: n = 3, m = 2, B = 450, T = 20 e F = R$50.000

	p1	p2	p3	L	C (R$)
m1	3	1	2	500	5000,00
m2	2	3	1	600	8000,00
b	0,25	0,15	0,10		
DMIN	500	600	400		
DMAX	1100	1200	800		
R (R$)	100,00	70,00	50,00		

Imagem 1: Exemplo de dados para o problema.

O programa deve funcionar para **qualquer conjunto de dados** (inclusive para qualquer *m* ou *n*) e deve fornecer a resposta completa: quanto vai ser produzido de cada produto, quantos lotes vão ser comprados de cada matéria-prima, quantas horas da linha estão sendo utilizadas (tanto na produção quanto nas trocas de produto), as receitas com cada produto e o lucro (ou prejuízo) total.

Modelagem

A modelagem deste problema iniciou pela definição dos nomes de cada constante e qual seria a sua finalidade, pegando como base as variáveis citadas durante o problema, temos que:

- *n: número de produtos fabricado;*
- *m: número de matérias-primas utilizadas;*
- *B: número total de horas mensais;*
- *T: quantidade de horas necessárias para a trocar o produto na linha de produção.*
- *F: custo fixo mensal.*
- R_i*: valor de venda de cada produto;*
- C_j*: preço de um lote;*

- L_j: *quantidade de matéria-prima por lote;*
- a_{ij}: *quantidade de matéria-prima utilizada por produto;*
- b_i: *quantidade de horas para o produto ser produzido*.

Após isso, foi iniciado a montagem da função objetiva, sendo assim primeiramente foi visto se o problema se trata de uma maximização ou minimização. Verificando bem o anunciado é notável a todo momento a necessidade de conseguir o maior lucro possível da fábrica, então este problema é de **maximização.** Na sequência, foram declaradas algumas variáveis, para ajudar neste problema, que nas quais são:

- p_i: *representa a quantidade de cada produto fabricado;*
- x_j: *representa a quantidade de lotes comprados, para cada matéria-prima*.
- z_i: *representa se o produto foi ou não fabricado*.

Com as variáveis declaradas, a função objetivo foi montada, sendo que ela possui duas partes, uma representando os ganhos da empresa, enquanto a outra representa as despesas mensais da empresa, dessa forma:

$$lucro = \sum_{i=1}^{n}(p_i * R_i) - (F + \sum_{j=1}^{m}(x_j * C_j))$$

Com a função objetiva criada, foi iniciada a criação das restrições, onde cada uma representa um limite para a solução objetiva, poder encontrar seu valor ótimo. Desse modo, as restrições para este modelo, são definidas logo abaixo:

Horas mensais

A primeira restrição se dá pela a utilização das horas mensais da fábrica, para isso foi utilizada a quantidade produtos e a quantidade de horas para ele ser produzido, além do tempo de troca de produção e o

tempo total de horas mensal. Fazendo um somatório, da quantidade de produto fabricado (***i***), seja multiplicado pela a quantidade de horas necessárias para a sua fabricação. Logo após, é feito um somatório verificando se o produto foi ou não fabricado (z_i), para que então ele seja multiplicado pelo valor de trocas por produto. Desta forma, a equação abaixo representa essa restrição.

$$\sum_{i=1}^{n}(p_i * b_i) + \sum_{i=1}^{n} z_i * T <= B$$

A primeira parcela dessa equação, representa a quantidade de horas que foi utilizada mensalmente, já na segunda parcela temos que a quantidade horas utilizadas deve de ser menor igual à quantidade de horas disponibilizadas pela fábrica.

Quantidade mínima de produtos

Os produtos da fabrica podem ou não ser produzidos, porém caso seja decidido a sua construção, deve existir uma quantidade mínima a ser respeitada. Para isso a equação abaixo foi montada, onde o z_i identifica se o produto foi ou não produzido.

$$p_i >= DMIN_i * z_i, \forall i = 1, ..., n$$

Quantidade máxima de produtos

Do mesmo jeito que a quantidade mínima, deve existir uma quantidade máxima de produtos a serem fabricados. Para definir essa restrição a equação abaixo foi criada, onde que o z_i identifica se o produto foi ou não produzido.

$$p_i <= DMAX_i * z_i, \forall i = 1, ..., n$$

Lotes de matérias-primas

Cada produto ao ser fabricado utiliza uma quantidade de matéria-prima de um lote, toda vez que este lote acaba, um novo lote é comprado, até

que as horas mensais sejam utilizadas. Sendo assim temos que a equação abaixo representa esta equação:

$$\sum_{i=1}^{n}(p_i * a_{ij}) <= x_j * L_j, \forall j = 1, ..., m$$

Restrição da não negatividade

Após do desenvolvimento de todas as restrições no modelo, foram criadas as restrições de não negatividade para as variáveis utilizadas, como podem ser visto abaixo:

- $p_i >= 0, \forall i = 1, ..., n$
- $x_j >= 0, \forall j = 1, ..., m$
- $0 <= z_i <= 1, \forall i = 1, ..., n$
- *z inteiro*.

Desenvolvimento

Após a modelagem do problema, iniciou-se a etapa da programação para resolver este modelo, dessa maneira foi utilizada a linguagem de programação ***python*** e para o ***solver*** foi utilizado o **SCIP**, encontrado na biblioteca `ortools.linear_solver` **,** distribuída pelo Google.

Iniciando a programação, foi definido como seria o arquivo de entrada com os dados do problema, para que estes dados sejam utilizados na execução do software. Para isso, foi definido a estrutura de como o arquivo será apresentado, então temos:

- Na primeira linha será definida a quantidade de produtos fabricados;
- Na segunda, temos o quantidade de matérias-primas utilizadas em sua produção;
- A terceira é inserida a quantidade de horas mensais da fábrica;
- A quarta linha está o valor da quantidade de horas para a troca de produto;
- Na quinta, tem o valor do custo fixo mensal;

- A partir da sexta são descritas as características do produto os produtos, ou seja, a quantidade de cada matéria-prima utilizada, o tempo de produção dele, a sua quantidade mínima e máxima se produzido e por fim o seu valor de venda;
- E após a descrição dos produto, são descritos as características da matéria prima, com a sua quantidade por lote e o custo para cada lote.

Se tomarmos o exemplo descrito na imagem 1, temos que a imagem 2 demonstra como são distribuídas em cada linha no arquivo de entrada e o script 1 demonstra como serão descritos estes valores no arquivo de entrada.

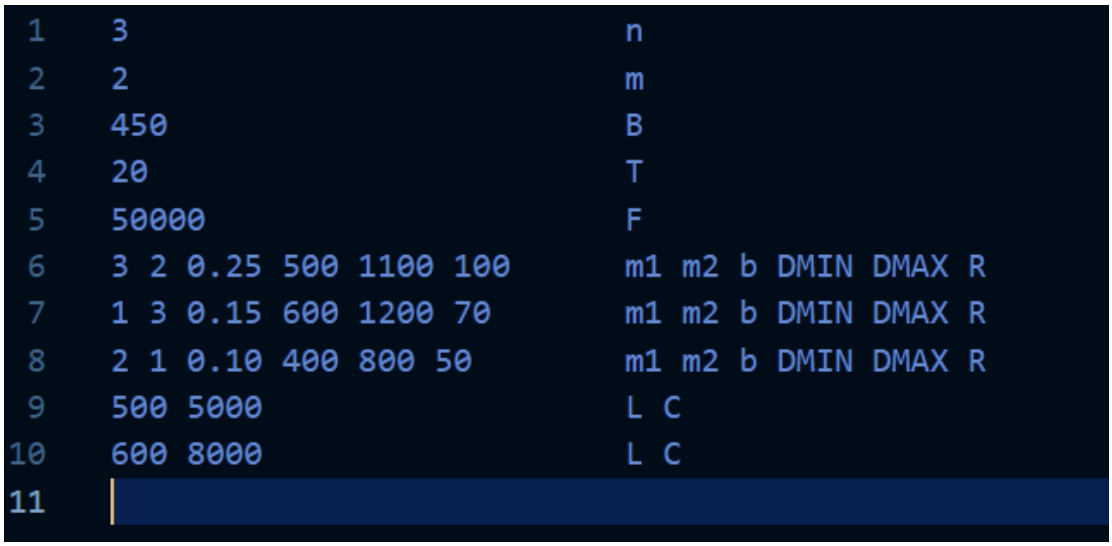

Imagem 2: Descrição do arquivo de entrada.

```
3
2
450
20
50000
3 2 0.25 500 1100 100
1 3 0.15 600 1200 70
2 1 0.10 400 800 50
500 5000
600 8000
```

Com a definição dos dados de entrada, foi criado um script para leitura do arquivo, com isso foram feitas as funções ***load_file*** **e** ***split_file.*** A primeira função abre o arquivo para a leitura, enquanto a outra lê o arquivo e atribui o seu valor designando para as constantes descritas na modelagem.

```
def load_file(self):
        self.data = open(f'./input/{self.filename}', 'r')
```

```
def split_file(self):
        lines = self.data.readlines()
        n = int(lines[0])
        m = int(lines[1])
        B = int(lines[2])
        T = int(lines[3])
        F = int(lines[4])

        products = []

        for i in range(5, 5 + n):
            p = lines[i].split(' ')
            product = {}

            for j in range(1, m + 1):
                product[f'm{j}'] = int(p[j - 1])

            product['b'] = float(p[m])
            product['DMIN'] = int(p[m + 1])
            product['DMAX'] = int(p[m + 2])
            product['R'] = int(p[m + 3])
            products.append(product)

        materials = []
        for i in range(5 + n, 5 + n + m):
            material = {}
            l = lines[i].split()
            material['lote'] = int(l[0])
            material['coast'] = int(l[1])
            materials.append(material)

        return n, m, B, T, F, products, materials
```

Com os valores das constantes obtidos, foi iniciado o *solver* **SCIP** que será utilizado nesse trabalho. Com isso, foram criadas as variáveis utilizando a função ***init_variables,*** na sua inicialização já foram definidas as restrições de não negatividade, como citadas na modelagem.

```
def init_variables(self):
        for i in range(1, self.n + 1):
            self.variables[f'p{i}'] = self.solver.IntVar(
                0.0, self.infinity, f'p{i}')

        for j in range(1, self.m + 1):
            self.variables[f'x{j}'] = self.solver.IntVar(
                1.0, self.infinity, f'x{j}'
            )

        for i in range(1, self.n + 1):
            self.variables[f'z{i}'] = self.solver.IntVar(
                0.0, 1.0, f'z{i}')
```

Após a inicialização das variáveis, foram criadas as restrições definidas na modelagem, sendo assim, cada restrição deve ser adicionada no ***solver*** e para isso a função ***create_restriction*** foi construída e utilizada**.**

```
def create_restriction(self):
    self.solver.Add(
       self.solver.Sum([self.variables[f'p{i}'] * self.products[i-1]['b']
                        for i in range(1, self.n + 1)])
         + self.solver.Sum([self.variables[f'z{i}']
                            for i in range(1, self.n + 1)]) * self.T <= self.
B
     )

    for i in range(1, self.n + 1):
       self.solver.Add(
          self.variables[f'p{i}'] >= self.products[i - 1]['DMIN'] * self.v
ariables[f'z{i}'])

    for i in range(1, self.n + 1):
       self.solver.Add(
          self.variables[f'p{i}'] <= self.products[i - 1]['DMAX'] * self.v
ariables[f'z{i}'])

    for j in range(1, self.m + 1):
       self.solver.Add(self.solver.Sum([self.variables[f'p{i}'] * self.pro
```

```
ducts[i - 1][f'm{j}'] for i in range( 1, self.n + 1)]) <= (self.variables
[f'x{j}'] * self.materials[j - 1]['lote']))
```

Na sequência foi criada a função ***set_function_objective***, com isso é dito ao ***solver*** que o problema é de maximização e a partir daí, é descrita a função objetiva do modelo vista na modelagem.

```
def set_function_objective(self):
        self.solver.Maximize(
            self.solver.Sum([self.variables[f'p{i}']*self.products[i - 1]
['R'] for i in range(1, self.n + 1)]) -
            (self.F + self.solver.Sum(
                [self.variables[f'x{j}'] * self.materials[j - 1]['coast']
 for j in range(1, self.m + 1)]))
        )
```

Por fim, foi criada uma função para mostrar o resultado das soluções encontradas, tornando a visualização do resultado muito mais clara e compreensível. A saída dessa função pode ser observada na imagem 3 .

```
Lucro = R$ 13759
Produtos:
  Foram produzidos 933 produtos p1, utilizou 233.25 horas na linha de producao, com a receita de R$ 93300.
  Foram produzidos 1178 produtos p2, utilizou 176.7 horas na linha de producao, com a receita de R$ 82460.
  Foram produzidos 0 produtos p3, utilizou 0.0 horas na linha de producao, com a receita de R$ 0.

Materias-primas:
Foram comprados 8.0 lotes da materia-prima m1, com o custo de R$ 40000.
Foram comprados 9.0 lotes da materia-prima m2, com o custo de R$ 72000.

A quantidade de horas que foram utilizadas na troca de produtos foi de 40.0 horas.
```

Imagem 3: Solução para o exemplo descrito no problema.