Otimização
Trabalho 1

Gustavo do Prado Silva
GRR20203942

SUMÁRIO

1. INTRODUÇÃO

O objetivo deste documento é relatar como foi produzido o Trabalho 1 da Disciplina de Otimização, expondo a seguir:

- O problema proposto;
- A modelagem usada para sua solução;
- O programa que gera a entrada para o `lp_solve`, que por sua vez executa o método Simplex.

Os tópicos serão abordados durante a seção de Desenvolvimento, tendo uma seção de conclusão para consolidar os resultados obtidos.

Todo o conteúdo que será apresentado a seguir baseou-se nas aulas da disciplina de Otimização, não recorrendo à autores específicos para a realização do trabalho, utilizando mais os conteúdos vistos em sala e materiais de apoio disponibilizados pelo professor em:

https://www.inf.ufpr.br/murilo/otimiza/otimiza.html

Relatório e trabalho produzidos por: Gustavo do Prado Silva - GRR20203942

2. DESENVOLVIMENTO

2.1 O Problema

2.1.1 Problema dado do enunciado

Uma empresa química produz n tipos diferentes de produtos. Para produzir esses produtos, usa diferentes proporções de m diferentes compostos (matérias-primas). Cada produto i tem valor de venda (por litro), v_i. Cada composto j usado tem um preço (por litro), p_j, e um limite diário de volume (em litros), q_j. A quantidade (em litros) de uso de cada composto j na produção de 1 litro do produto i é dada por $c_{i,j}$. Ou seja, para produzir 1 litro do produto i são necessários $c_{i,1}$ litros do composto 1, $c_{i,2}$ litros do composto 2, e assim até $c_{i,m}$ litros do composto m. A empresa assume que toda sua produção será vendida. Levando em consideração os dados, e que os demais custos de produção não dependem de quais produtos são feitos, a empresa quer maximizar os lucros.

2.1.2 Introdução ao problema

O problema tratado por este trabalho refere-se a um Problema Linear (PL) de otimização cuja instância é resolvida pelo algoritmo simplex. O problema nos descreve uma empresa química, que quer maximizar os lucros da venda de produtos químicos. Temos que cada um dos produtos tem como matéria-prima m compostos diferentes, porém cada composto possui um limite de produção diário (em litros). O problema nos deixa assumir que toda a produção é vendida, ou seja, não há estoque dos produtos produzidos. Para modelar o problema, seguiremos a seguinte abordagem:

- Pensaremos primeiro na função objetivo, seguindo a definição intuitiva de lucro: Tudo que recebo menos tudo que gasto, ou seja, o valor total de venda dos n produtos <u>menos</u> os custos de produção dos m compostos.

- Para as equações das restrições, usaremos os limites de produção diários dos compostos como base para nossas inequações, visto que a quantidade total de litros de compostos utilizada necessita ser <u>no máximo</u> igual aos limites diários.

2.2 A Modelagem

2.2.1 Função Objetivo

Primeiramente, a função objetivo: queremos maximizar os lucros da empresa. Para ter o lucro dado por cada produto, temos que considerar o custo de produção de cada composto que o forma e o valor por qual cada produto é vendido.
Seja:

- p_j: o custo (por litro) do composto j
- p_i: o custo (por litro) do produto i
- v_i: o valor de venda do litro do produto i
- $c_{i,j}$: quantidade (em litros) do composto j no produto i

Temos que o custo de cada produto é a soma da quantidade de todos os compostos que o compõem multiplicada pelo custo de cada composto, ou seja:

$$p_i = \sum_{j=1}^{m} p_j \times c_{i,j}$$

E o lucro gerado da venda dos produtos é o valor arrecadado pela venda de cada um, menos os custos de produção do mesmo. Como queremos maximizar os lucros, temos que a função objetivo corresponde a: **max:** $\sum_{i=1}^{n} v_i - p_i$

2.2.2 Equações

Agora, as equações que modelam o problema.
Temos que respeitar a quantidade limite de cada composto em cada um dos produtos, logo a soma das quantidades do composto j em cada produto i tem de ser menor ou igual ao limite do composto j. Por exemplo:

Portanto:

- Seja $c_{i,j}$ a quantidade do composto j no produto i
- Seja x_i a quantidade, em litros, do produto i
- Seja q_j o limite de produção do composto j

Temos então, m equações da forma

$$\sum_{i=1}^{n} c_{i,j} \times x_i \leq q_j$$

Sendo uma para cada composto j, ou seja:

$$c_{1,j} * x_1 + c_{2,j} * x_2 + ... + c_{n,j} * x_n \leq q_j$$

$$c_{1,j+1} * x_1 + c_{2,j+1} * x_2 + ... + c_{n,j} * x_n \leq q_{j+1}$$

$$...$$

$$c_{1,m} * x_1 + c_{2,m} * x_2 + ... + c_{n,j} * x_n \leq q_m$$

São as equações que restringem o espaço de busca do problema, juntamente com as restrições de que $x_i \geq 0$ para i = 0, 1, ..., n

2.3 A Solução

2.3.1 `lp_solve`

O pacote linux `lp_solve` pode ser instalado em sistemas Ubuntu utilizando:

```
sudo apt install lp-solve
```

Ele resolve Problemas de Otimização usando o método Simplex. As entradas podem ser lidas da entrada padrão (STDIN) ou de arquivos LP ou MPS, gerando um resultado na saída padrão (STDOUT).

2.3.2 Arquivos

- `Makefile`: Makefile para gerar o executável "producao"e o arquivo PDF ".pdf"
- **exemplos/**
  - exemplo.in: Exemplo de entrada dado no texto do enunciado.
  - exemplo1.lp: Resolução do problema usando a modelagem proposta, utilizando o algoritmo .
  - exemplo2.lp: Exemplo de entrada para o `lp_solve`, dado no enunciado e usado para testar o programa `lp_solve`.
- Relatorio-gps20.pdf: Documento PDF deste relátorio.
- `producao.c`: Arquivo fonte em C usado para tratar a entrada do problema.

2.3.3 Uso do Programa

Para gerar o arquivo executável do programa, é necessário digitar o comando `make` no terminal, que gerará o arquivo `producao`. Ao ser executado, o programa lê a entrada da `STDIN` e gera a saída correspondente em `STDOUT`, com as equações e função objetivo que serão utilizadas no `lp_solve`.

Porém, também é possível redirecionar a entrada de um arquivo para o programa, como por exemplo: `./producao < exemplo.txt`

Analogamente, também é possível redirecionar a saída para um arquivo, como por exemplo: `./producao > saida.txt`

Ou juntando ambos: `./producao < entrada.txt > saida.txt`

É possível também utilizar a saída do programa `producao` diretamente como entrada do `lp_solve` utilizando os pipes do linux, como por exemplo:

```
./producao < entrada.txt | lp_solve
```

2.3.4 Exemplo de entrada do enunciado

No enunciado temos o seguinte exemplo de entrada para o programa:

3 4
10 7 3
1 1000
2 2000
5 500
10 2000
0.2 0.5 1.0 0.1
1.0 0.1 0.3 0.1
0.4 0.2 0.2 0.0

Ao utilizar a modelagem e algoritmo propostos temos a seguinte saída para o `lp_solve`:

max: 2.8x0 + 3.3x1 + 1.2x2;
0.2x0 + 1x1 + 0.4x2 <= 1000;
0.5x0 + 0.1x1 + 0.2x2 <= 2000;
1x0 + 0.3x1 + 0.2x2 <= 500;
0.1x0 + 0.1x1 + 0x2 <= 2000;

Tal resultado, ao ser usado como entrada para o `lp_solve`, gera a seguinte saída:

Value of objective function: 3755.31914894
Actual values of the variables:
x1 212.766
x2 957.447
x3 0

Ou seja, temos o resultado ótimo como `R$` 3755,31, produzindo 212.766 litros do produto 1, 957.447 litros do produto 2 e 0 litros do produto 3, que é o resultado esperado no enunciado do exemplo.

2.3.5 Programa em C

```c
int n, m;
scanf("%d", &n);
scanf("%d", &m);

// Alocação dos vetores para guardar os valores
int *preco  = malloc(sizeof(int) * n);
double *lucro  = malloc(sizeof(int) * n);
int *custo   = malloc(sizeof(int) * m);
int *limite  = malloc(sizeof(int) * m);

// Matriz n x m com quantidades dos produtos/compostro
double **mat = malloc(sizeof(double*) * m);
for(int i = 0; i < m; ++i){
   mat[i] = malloc(sizeof(double*) * n);
}
```

Aqui temos as principais variáveis e estruturas de dados do programa:

- As varáveis que armazenam n e m
- Os vetores com valores de custo, lucro, limite e preço
- A matriz com as quantidades dos compostos em cada produto, onde cada linha representa um composto e cada coluna um produto, ou seja, tem m linhas por n colunas

```c
/*
   Lê valores com os preços
*/
for(int i = 0; i < n; ++i){
   scanf("%d", &(preco[i]));
}


/*
   Lê valores de custo e limite de produção
*/
for(int i = 0; i < m; ++i){
   scanf("%d", &(custo[i]));
   scanf("%d", &(limite[i]));
}


/*
   Lê matriz com os coeficientes
*/
for(int i = 0; i < n; ++i){
   for(int j = 0; j < m; ++j){
      scanf("%lf", &(mat[j][i]));
   }
}
```

Aqui estamos apenas lendo da entrada padrão os valores de entrada do problema e armazenando nas estruturas.

```
/*
   Calcula lucro baseado: preco(x0) - custo(x0)*quantidade(x0)
*/
for(int i = 0; i < n; ++i){
   double custoTotali = 0.0;
   for(int j = 0; j < m; ++j){
      custoTotali += custo[j] * mat[j][i];
   }
   lucro[i] = preco[i] - custoTotali;
}
```

Aqui estamos calculando o somatório correspondente a função objetivo, ou seja, a função que calcula o lucro total da venda dos produtos.

```
/*
   Imprime matriz de coeficientes já como variaveis para o lp_solve
*/
for(int i = 0; i < m; ++i){
   printf("%5gx%d", mat[i][0], 0);
   for(int j = 1; j < n; ++j){
      printf("   + %5gx%d", mat[i][j], j);
   }
   printf("  <= %5d;\n", limite[i]);
}

for(int j = 0; j < n; ++j){
   printf("x%d >= 0;\n", j);
}
```

Aqui estamos imprimindo na saída padrão as equações de restrição, usando a notação de: numx_0 até numx_{n-1}

```
/*
   Libera todas as estruturas
*/
free(preco);
free(custo);
free(limite);
for(int i = 0; i < n; ++i){
   free(mat[i]);
}
free(mat);
```

Por fim, liberamos todas as estruturas utilizadas e encerramos o programa

3. CONCLUSÃO

Tendo em vista a modelagem e a solução, juntamente com seus exemplos, podemos observar que a modelagem proposta para a resolução do problema é efetiva no exemplo dado.
Isso ocorre pois na mesma estamos considerando o lucro como:

$$\text{Lucro} = \text{Arrecadações - Custos}$$

o que corresponde não apenas à realidade como ao contexto do problema em si. Também podemos traçar um paralelo com as equações de restrição, onde a quantidade máxima de uma produto que pode ser vendida é aquela que foi produzida, ou seja:

$$\text{Quantidade disponível} \leq \text{Limite de produção}$$

Dessa forma, podemos concluir que dada modelagem e algoritmo de solução são adequados para a resolução do problema proposto, gerando as respostas desejadas.