Otimização - Relatório T1

Nico I. G. Ramos

GRR20210574

Abril 2023

1 Introdução

Problemas de programação linear podem ser descritos como problemas nos quais se busca maximizar(ou minimizar) uma função linear, chamada de função objetivo, limitada por restrições lineares.

Esse problemas podem ser resolvidos pelo algoritmo **Simplex**. Esse algoritmo é monotônico em relação à função objetivo, pois em cada iteração o valor dela nunca decresce. Isso se dá pois como a solução ótima estará em um dos vértices do politopo formado pelas restrições, o **Simplex** percorre esses vértices indo, a cada vértice, para o vértice vizinho que mais faça a função objetivo crescer.

O objetivo desse trabalho é produzir uma entrada para o programa **lp_solve**, que implementa o algoritmo do **Simplex**, com uma modelagem para o problema da *produção de produtos químicos*.

2 Problema

O problema da *produção de produtos químicos* consiste em maximizar o lucro de uma empresa na produção de certos produtos químicos.

Cada produto possui um valor de venda e necessita de uma quantia de diferentes compostos para serem produzidos. Estes, por sua vez, possuem um valor de custo e um limite diário de uso. Além disso, a empresa assume que terá toda a sua produção vendida e que os demais custos não dependem de quais produtos são produzidos.

Como nesse problema busca-se maximizar a função lucro restringida pelos limites diários de cada composto, e tanto a função objetivo quanto as restrições são lineares, ele é uma instância da programação linear e pode ser resolvido pelo algoritmo **Simplex**.

3 Modelagem

Como o problema a ser modelado é um problema de maximizar o lucro de uma empresa, primeiro foi feita uma função para calcular o custo de produção de cada produto, em seguida a função que calcula o lucro de cada produto e, por fim, a função objetivo e as expressões que a limitam.

Seja m a quantidade de tipos de compostos utilizados pela empresa, c_{ij} a quantidade do composto j no produto i e m_j o custo do composto j, o custo $C(i)$ de um produto i pode ser calculado pela seguinte expressão:

$$C(i) = \sum_{j=1}^{m}(c_{ij} \times m_j)$$

Com isso, seja v_i o valor de venda do produto i, o lucro $L(i)$ de um produto i pode ser calculado pela seguinte expressão:

$$L(i) = v_i - C(i)$$

E, seja n a quantidade de produtos produzidos pela empresa e n_i a quantidade produzida do produto i, a função objetivo F pode ser obtida pela seguinte expressão:

$$F = \sum_{i=1}^{n}(L(i) \times n_i)$$

Ou, ainda,

$$\begin{aligned} F &= \sum_{i=1}^{n}(L(i) \times n_i) \\ &= \sum_{i=1}^{n}((v_i - C(i)) \times n_i) \\ &= \sum_{i=1}^{n}((v_i - \sum_{j=1}^{m} c_{ij} \times m_j) \times n_i) \end{aligned}$$

Por fim, seja q_j o limite diário do composto j, as expressões que limitam a função objetivo podem ser expressas da

seguinte maneira:

$$\forall\ i \in \{1, .., n\},\ n_i \geq 0$$
$$\forall\ j \in \{1, .., m\},\ \sum_{i=1}^{n} c_{ij} \leq q_j$$

Portanto, o problema pode ser expresso como o seguinte problema linear:

$$\text{Max } \sum_{i=1}^{n}(L(i) \times n_i)$$
$$\text{S.A.}$$
$$L(i) = v_i - C(i)$$
$$C(i) = \sum_{j=1}^{m}(c_{ij} \times m_j)$$
$$\forall\ i \in \{1, .., n\},\ n_i \geq 0$$
$$\forall\ j \in \{1, .., m\},\ \sum_{i=1}^{n} c_{ij} \leq q_j$$

4 Implementação

A implementação pode ser dividida em três partes principais: a biblioteca de ponto fixo, a leitura do valores de entrada e a geração da saída.

A escolha por ponto fixo se deu pois apenas a quantidade de produtos produzidos e a quantidade de tipos de compostos são garantidamente inteiros.

4.1 Biblioteca de ponto fixo

Essa biblioteca implementa a conversão de uma *string* para um número em ponto fixo e a impressão de um número em ponto fixo. Para o problema proposto, foi escolhido utilizar três casas decimais para a precisão.

Ela também implementa a multiplicação de dois valores em ponto fixo, na qual o valor obtido na multiplicação é dividido por $10^{\text{precisão}}$.

4.1.1 Leitura

Como a parte inteira e a parte decimal são separadas por um ponto, é lido a parte antes do ponto, a parte inteira, seguida da parte depois do ponto, a parte decimal. Caso não seja encontrado um ponto, o número é inteiro e, portanto, não há nescessidade de processar a parte decimal.

- **Parte inteira:** para cada caractere na *string*, o resultado até aquele momento é multiplicado por 10 e somado do valor numérico correspondente ao caractere. Caso a *string* contenha um número ímpar de sinais de menos o número é negativo, caso contrário é positivo.
- **Parde decimal:** o valor é calculado da mesma maneira, mas os caracteres são lidos apenas até a precisão desejada e o valor é sempre positivo. Caso a *string* contenha menos casas decimais do que a precisão, o valor calculado é multiplicado por 10 até a precisão correta.

Por fim, caso a parte inteira seja positiva, a parte decimal é acrescida a ela, caso contrário, a parte decimal é subtraída. O que garante que tanto números positivos quanto negativos são convertidos corretamente.

4.1.2 Impressão

Primeiro é impressa a parte inteira, seguida do ponto e, por último, a parte decimal.

- **Parte inteira:** obtida pela divisão do número por $10^{\text{precisão}}$.
- **Parte decimal:** primeiro o número é dividido pela potência de dez correspondente e em seguida o resultado é divido pela potência de dez anterior, o valor obtido é um dos dígitos. Isso é feito para cada dígito, do mais significativo para o menos.

Como a biblioteca não verifica o sinal da impressão, o número precisa ser positivo, caso contrário a impressão terá o sinal de menos tanto na parte positiva quanto na parte negativa.

4.2 Leitura da entrada

Para a leitura, cada valor é lido e é verificado se não é um espaço, uma tabulação ou uma quebra de linha, caso não seja, é adicionado ao final do vetor de leituras.

Com isso, é possível ler os valores independente da formatação da entrada, pois ao final da leitura, basta processar o vetor e obter os valores numéricos.

4.2.1 Processamento

Primeiro é processado o total de produtos e de compostos, caso um deles não seja maior que zero, o programa encerra sem gerar nenhuma saída, pois se não tiver nenhum

produto ou os produtos não forem compostos por nada não há o que ser maximizado.

Em seguida é feito o processamento dos valores de venda de cada produto, seguido do processamento dos pares de *custo/limite diário* para cada composto. Por fim, a composição de cada produto é processada.

Dessa maneira, a implementação utiliza: dois inteiros, um para representar a quantidade de produtos e outro para a quantidade de tipos de compostos; três vetores de ponto fixo; um para o valor de venda de cada produto, um para o custo de cada composto e um com os respectivos limites diários; e um vetor de vetores de ponto fixo, que representa cada produto e a sua composição.

4.3 Geração da saída

A saída é gerada em duas partes: a função objetivo e as restrições.

Como a impressão de ponto fixo imprime apenas números positivos, primeiro é impresso o sinal do número e em seguida o módulo dele. O que garante que não será impresso um sinal de positivo seguido de um negativo na expansão dos somatórios e que a impressão de números negativos terá o sinal apenas na frente da parte inteira.

4.3.1 Função objetivo

É calculado a função lucro de cada produto, de acordo com a expressão obtida na modelagem, e concatenando cada valor obtido com um sinal de mais ou menos de acordo com o sinal dele. Cada valor é seguido por um x e pelo seu índice mais um.

4.3.2 Restrições

Para cada composto, é impresso a quantidade usada dele em cada produto, concatenados de acordo com o sinal, seguido de um x e do índice do produto mais um. Por fim, o operando de menor e o limite diário daquele composto.

As restrições de não-negatividade não são impressas pois elas são definidas por padrão no **lp_solve** e é impresso somente o sinal de menor, e não o de menor igual, pois não há diferença entre os dois no *lp_solve*.

5 Resultados

Para verificar se o algoritmo gera saídas corretas, os testes foram escolhidos pensando na leitura e processamento dos valores, na implementação do ponto fixo e na impressão dos somatórios expandidos. Dessa maneira, os testes tiveram por objetivo assegurar a impressão correta dos valores independente da formatação e da precisão na entrada e independente do sinal.

Os testes realizados podem ser encontrados no diretório **exemplos** e executados com o comando **source ./teste.sh exemplos producao**.

5.1 Precisão

Para verificar se os números na saída gerada tem a precisão correta, foi testada uma entrada com valores com nenhuma casa decimal e com mais casas decimais do que a precisão.

O *Exemplo 1* e o *Exemplo 2* ilustram, respectivamente, a entrada recebida, e a saída gerada pelo programa.

```
1 1
10.123456789
1 1000
1
```

Exemplo 1: Entrada com mais casas decimais do que a precisão.

```
max: 9.123x1;
1.000x1 < 1000.000;
```

Exemplo 2: Saída para a entrada com mais casas decimais do que a precisão.

5.2 Zero

Para verificar se o zero é impresso de maneira, foi gerada uma entrada na qual a função objetivo é nula. A saída pode ser observada no *Exemplo 3*.

```
max: 0.000x1;
-10.123x1 < 1000.000;
```

Exemplo 3: Saída para função objetivo com valor nulo

5.3 Sinais

Para assegurar que os sinais são impressos corretamente, foi verificado a impressão do sinal na função objetivo e nas restrições, como mostra o *Exemplo 4* e o *Exemplo 5*, respectivamente.

```
max: 9.000x1 - 9.000x2 + 9.000x3;
1.000x1 + 0.000x2 + 1.000x3 < 1000.000;
0.000x1 + 1.000x2 + 0.000x3 < 1000.000;
```

Exemplo 4: Saída para função objetivo com sinais misturados

```
max: 19.999x1;
-9.999x1 < 1000.000;
```

Exemplo 5: Saída para restrição com valor negativo

5.4 Índice

Para verificar se índices com mais de um dígito são impressos de maneira correta, uma entrada com dez produtos foi gerada e a saída é mostrada no *Exemplo 6*.

```
max: 9.000x1 + 9.000x2 + 9.000x3 + 9.000x4 + 9.000x5 + 9.000x6 + 9.000x7 + 9.000x8 + 9.000x9 + 9.000x10;
1.000x1 + 1.000x2 + 1.000x3 + 1.000x4 + 1.000x5 + 1.000x6 + 1.000x7 + 1.000x8 + 1.000x9 + 1.000x10 < 1000.000;
```

Exemplo 6: Saída com mais dez produtos

5.5 Entrada não formatada

Como a entrada não tem um formato padrão, foi gerada uma entrada com os valores espaçados por uma combinação de múltiplos espaços, tabulações e novas linhas. O *Exemplo 7* mostra a entrada, no *Exemplo 8*, é possível observar que apesar do espaçamento não formatado, a saída gerada é a correta.

As únicas restrições são que os valores devem aparecer na ordem correta e que todas as componentes de um mesmo número devem aparecer sequencialmente, sem nenhum tipo de espaço entre elas.

```
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
```

Exemplo 7: Entrada não formatada

```
max: 2.800x1 + 3.300x2 + 1.200x3;
0.200x1 + 1.000x2 + 0.400x3 < 1000.000;
0.500x1 + 0.100x2 + 0.200x3 < 2000.000;
1.000x1 + 0.300x2 + 0.200x3 < 500.000;
0.100x1 + 0.100x2 + 0.000x3 < 2000.000;
```

Exemplo 8: Saída para entrada não formatada

6 Conclusão

Através dos testes realizados é possível perceber que modelagem e a implementação da geram entradas adequadas para o **lp_solve**.

A modelagem formula, através de expressões lineares, a função objetivo e as restrições. Ela considera os custos de produção de cada produto, os valores de venda e os limites diários de uso dos compostos.

Por sua vez, a implementação garante que os valores serão impressos corretamente independente do espaçamento e da precisão na entrada, do sinal e da quantidade de produtos ou tipos de compostos.