Relatório de Projeto Supervisionado MS777
Problema do Troco: uma resolução via programação dinâmica

Orientadora: Kelly Cristina Poldi - IMECC/UNICAMP
Aluno: Gustavo Leite Machado - RA: 169356 - IMECC/UNICAMP

1º Semestre de 2018

Resumo

Neste trabalho apresentamos um problema clássico de Otimização Combinatória e uma técnica muito útil para a resolução de alguns tipos de problemas. Para isso, estudamos como solucionar, via programação dinâmica, o problema do troco. Esse problema consiste em encontrar as possíveis formas de selecionar recursos disponíveis, moedas, de modo a utilizarmos o menor número possível deles para atingirmos um determinado valor, o troco. Uma forma de ilustrar este problema é uma transação financeira, como a realização de compras no mercado, por exemplo.

1 Introdução

Em muitos casos, uma forma de resolver um problema é buscar aproximá-lo de algum outro problema mais simples que já saibamos como resolver e temos maior familiaridade. Contudo, nem sempre é possível fazer isso de maneira ideal, sem perda de generalidade ou grandes alterações na solução. Porém, quando se é, de fato possível realizar essa aproximação e percebemos que se trata de um problema de otimização combinatória, a programação dinâmica é um dos métodos que podem ser aplicados na resolução do problema.

A programação dinâmica é útil para a resolução de problemas que podem ser fragmentados como sequência de subproblemas menores e mais fáceis de serem resolvidos, que quando sobrepostos compõem o problema original, de tal forma que uma solução ótima seja computada a partir das soluções ótimas dos subproblemas (Arenales *et. al* [1]; Cormen *et. al* [2]; Dasgupta *et. al* [3]).

Um problema clássico no estudo de programação dinâmica é o problema do troco, Wright [4]. Vamos analisar neste trabalho o problema do troco buscando encontrar uma solução que tenha o menor uso possível de moedas. Outras

possíveis análises desse problema poderiam ser quantas soluções possíveis para dar o troco temos disponíveis e também, quais são essas soluções.

2 Programaçao dinâmica

Antes de definirmos com mais detalhes o problema do troco, faremos uma breve apresentação da técnica de programação dinâmica. Para isso, iniciamos apresentando um exemplo ilustrativo.

2.1 Exemplo

Considere o seguinte exemplo (adaptado de Arenales *et. al* [1]): Temos uma empresa I, fabricante de um determinado produto P. Sabemos que atualmente a empresa não tem deste produto em estoque, e que a demanda por esse produto nos próximos 5 meses ($t = 1, 2, 3, 4, 5$) é de 30, 20, 60, 10 e 40 toneladas, respectivamente.

O custo de produção do produto P é de 5 mil unidades monetárias por tonelada e o custo de armazenamento é de 20% do custo de produção do produto armazenado. Sabemos também que a máquina que produz o produto tem um custo fixo de 10 mil unidades monetárias para sua preparação, e que ela só produz em lotes múltiplos de 10 toneladas, ou seja, ela consegue produzir 10, 20, 30 toneladas e assim por diante. Qual o plano de produção que minimiza os gastos da empresa para esses cinco meses?

Para esse problema vamos começar analisando o último mês, $t = 5$. Como manter qualquer estoque tem um custo, sabemos que neste mês em particular não queremos nenhum estoque após a demanda ser atendida, entretanto, nada impede que em $t = 4$ tenha-se feito um estoque para ser utilizado em $t = 5$. Assim, começamos a analisar as possibilidades de produção; ou não temos estoque do mês anterior ou temos algum estoque do mês anterior. Ou seja, a quantidade do produto P a ser produzida depende do quanto temos de estoque no início do mês.

O objetivo agora é encontrarmos o melhor plano de produção para o quinto mês. Uma vez definido esse plano conseguimos saber o estoque final do quarto mês. E assim conseguimos agora definir o melhor plano de produção para o quarto mês, que somando ao plano definido para o quinto mês nos dê o menor custo de produção para esses dois meses. Consequentemente, conseguimos definir também o seu estoque inicial. Repetimos esse processo para os outros meses, até o primeiro, onde sabemos que o estoque inicial é zero. Note que para definirmos a melhor decisão do mês i precisamos saber o plano ótimo do mês $i + 1$ até o final do período analisado.

Podemos afirmar então que o estoque ao final do mês $i - 1$, e_{i-1}, influencia a decisão do quanto se deve produzir no mês i, $d_i(e_{i-1})$. Dessa decisão obtemos um custo $c_i(e_{i-1}, d_i(e_{i-1}))$ que depende do estoque inicial do mês em questão e do quanto foi produzido no mês anterior, e um estoque no final do mês i.

2.1.1 Cálculos do exemplo

Cálculos para o quinto mês ($t = 5$)

Sabemos que ao final do mês $t = 5$ a empresa não quer nenhum estoque do produto, logo, $e_5 = 0$. Temos agora que analisar os possíveis custos que ela pode vir a ter, variando o estoque inicial, e_4. Assim:

- Supondo estoque incial igual a zero ($e_4 = 0$).

  $d_5(e_4) = d_5(0) = 40 \Rightarrow c_5(0, 40) = 40 \times 5 + 10 = 210$ mil

- $e_4 = 10$

  $d_5(e_4) = d_5(10) = 30 \Rightarrow c_5(0, 30) = 30 \times 5 + 10 = 160$ mil

- $e_4 = 20$

  $d_5(e_4) = d_5(20) = 20 \Rightarrow c_5(0, 20) = 20 \times 5 + 10 = 110$ mil

- $e_4 = 30$

  $d_5(e_4) = d_5(10) = 10 \Rightarrow c_5(0, 10) = 10 \times 5 + 10 = 60$ mil

- $e_4 = 40$

  $d_5(e_4) = d_5(40) = 0 \Rightarrow c_5(0, 0) = 0$

Portanto, temos que o menor custo possível de produção no quinto mês seria se toda a demanda fosse suprida pelo estoque inicial, ou seja, não se produzisse nada.

Cálculos para o quarto mês ($t = 4$)
Sabemos, pelo que encontramos resolvendo o problema para $t = 5$, que o estoque final deste mês é 40 toneladas do produto P, pois vimos que a melhor solução seria obtida no caso em que a empresa não produz nada no quinto mês, e o gasto de estoque é: $40 \times 5 \times 0,1 = 20$ mil.

Como sabemos a demanda do mês e seu estoque final, conseguimos relacionar a produção, $d_4(e_3)$, e o estoque inicial e_3.

Relembrando: $e_i + d_{i+1}(e_i) -$ demanda do mês $i + 1 = e_{i+1}$

Assim, sabemos que $e_3 + d_4(e_3) = 50$. Ou seja, estamos utilizando o resultado da solução do subproblema para o mês 5 para nos ajudar a resolver para o mês 4. E essa é a essência da programação dinâmica. Utilizar resultados de subproblemas anteriores na resolução do subproblema seguinte.

- $e_3 = 0 \Rightarrow d_4(0) = 50$

  $c_4(0, 50) = 50 \times 5 + 10 + 20 = 280$ mil

- $e_3 = 10 \Rightarrow d_4(0) = 40$

  $c_4(10, 40) = 40 \times 5 + 10 + 20 = 230$ mil

- $e_3 = 20 \Rightarrow d_4(0) = 30$

  $c_4(20, 30) = 30 \times 5 + 10 + 20 = 180$ mil

- $e_3 = 30 \Rightarrow d_4(0) = 20$

  $c_4(30, 20) = 20 \times 5 + 10 + 20 = 130$ mil

- $e_3 = 40 \Rightarrow d_4(0) = 10$

  $c_4(40, 10) = 10 \times 5 + 10 + 20 = 80$ mil

- $e_3 = 50 \Rightarrow d_4(0) = 0$

  $c_4(50, 0) = 20 = 20$ mil

Temos assim que a melhor decisão para este mês também é não produzir nada e suprir sua demanda com o estoque final do mês anterior.

Repetimos esse processo até o primeiro mês ($t = 1$). Fazendo isso obtemos que o melhor plano de ação para a empresa é produzir 160 toneladas do produto P no primeiro mês e estocar todo o excedente para suprir a demanda nos próximos meses. Claramente não estamos considerando que o produto possa estragar, por exemplo. Isso certamente mudaria o melhor plano de ação a ser adotado pela empresa e a forma como resolveríamos o problema.

2.2 Conceitos básicos da programação dinâmica

Analisando o exemplo apresentado na seção conseguimos ilustrar alguns conceitos básicos da programação dinâmica (Arenales *et. al* [1]; Cormen *et. al* [2]).

Para resolver o problemas nós o "quebramos"em problemas menores, que no exemplo são definidos pelos meses analisados, que resolvemos em sequência, um de cada vez, e a solução de cada subproblema nos dava informações para rosolver os subproblemas seguintes. Aqui temos o conceito de *estágios*.

Em cada mês analisado, para decidir se a fábrica produziria ou não naquele mês sempre analisávamos o estoque futuro (que ficará para o mês seguinte) e passado (resultado do mês anterior, quando possível). O estoque era a informação necessária para a tomada de decisão, e como vimos no exemplo, tínhamos diversos níveis de estoque disponível. Esse é o conceito de *estados*. Como podemos notar, em um *estágio* é possível termos diversos *estados*, que são as informações necessárias para a tomada de decisão.

Em cada *estágio* do exemplo tínhamos que tomar a *decisão* do quanto produzir, e essa decisão dependia do *estado* que foi observado (a quantidade a ser

produzida dependia do estoque), e mais, cada *decisão* tinha um impacto no mês em que ela foi tomada e no mês seguinte. As possíveis decisões no nosso exemplo são os tamanhos dos lotes do produto P.

Finalmente, chamamos a sequência de decisões tomadas para resolver o problema de *política*. Uma *política ótima* seria aquela que otimiza o objetivo desejado. Similar a soluções factíveis e solução ótima.

Um outro conceito importante, mas que é na verdade um princípio, é o *princípio de otimalidade de Bellman* (Arenales *et. al* [1]), que diz:

> "Em um conjunto de decisões ótimas, tem-se a propriedade de que, qualquer que seja a primeira decisão e o estado inicial, as decisões subsequentes têm de ser ótimas om respeito ao estado resultante dessa primeira decisão."

Esse princípio garante que podemos resolver os subproblemas e montar a partir de suas soluções ótimas a solução ótima do problema mais complexo.

3 O Problema do troco mínimo

3.1 Descrição do problema

O problema do troco mínimo é um problema clássico da programação dinâmica. Estamos falando em mínimo, pois queremos encontrar o menor número possível de moedas a serem utilizadas e não as possíveis formas de se dar o troco.

Assim, dado um conjunto finito $S = \{m_1, m_2, \ldots, m_k\}$ das k moedas que temos a disposição, todas com valores diferentes, precisamos retornar um troco N a um cliente.

Podemos aprofundar o estudo do problema do troco ao considerarmos, por exemplo, as seguintes restrições: temos um número limitado de moedas de mesmo valor, o qual varia de moeda para moeda; queremos utilizar mais uma certa moeda do que outra, etc. Neste estudo vamos considerar que temos um número inifinito de cada moeda do conjunto de moedas disponíveis e que não favorecemos nenhuma em relação as outras. Estamos, a princípio, interessados apenas em definir o menor número de moedas a serem utilizadas para dar o troco.

3.1.1 Possível aplicação

Suponha um caixa de mercado automatizado, o qual dispensa a necessidade de um operador para validar os produtos comprados, receber o pagamento, seja em cartão ou dinheiro e devolver o troco ao cliente quando necessário. É aceitável pensar que quanto mais tempo conseguirmos manter esse caixa operando sem parar para repor dinheiro de troco nele, melhor. Neste contexto podemos utilizar o problema do troco mínimo, onde buscamos devolver o troco ao cliente utilizando o menor número possível de moedas, para termos moedas para troco

dentro do caixa o maior tempo possível. Essa é uma visão simplificada do problema, que na prática provavelmente é muito mais complexo, pois temos um valor finito de moedas para troco, muito provavelmete deve ser feita uma priorização de quais moedas usar antes de outras, algumas moedas podem acabar antes que as outras, etc.

3.2 Formulação matemática do problema

A seguir, apresentamos um modelo matemático de programação linear inteira para o problema do troco.

Considere:
N: Um número inteiro que representa o valor do troco.
m_i: O valor da moeda do tipo i no conjunto de moedas disponíveis.
x_i: Variável inteira que indica o número de vezes que utilizamos a moeda de valor m_i.
k: A quantidade de tipos de moedas disponíveis.

Assim, o problema do troco pode ser formulado matematicamente como:

$$\text{minimizar } f(x) = \sum_{i=1}^{k} x_i$$

sujeita a:

$$\sum_{i=1}^{k} x_i m_i = N$$
$$x \in \mathbb{Z}^+$$

A função objetivo a ser minimizada representa o número de moedas utilizadas. A primeira restrição garante que o valor do troco seja atingido, enquanto a segunda garante que a variável x_i assume valores inteiros e positivos.

3.2.1 Uma possível forma de visualizar o problema

Para melhor visualizar como encontrar uma solução vamos imaginar uma árvore de decisão para o problema. Como já foi dito na descrição do problema, vamos considerar que temos um conjunto $S = \{m_1, m_2, \ldots, m_k\}$ das k distintas moedas disponíveis e precisamos devolver um troco de valor N a um cliente.

Inicialmte temos um problema inicial T(N, S) que vamos quebrar em dois, sendo esses dois ramos da árvore de decisão. Em um deles vamos obter o problema T(N-m_i, S), ou seja, subtraímos do troco o valor da moeda m_i, e mantemos o conjunto de moedas disponíveis sem fazer nenhuma alteração. O outro problema que obtemos é T(N, S-$\{m_i\}$), ou seja, mantemos o valor do troco a ser dado e removemos a moeda i do conjunto de moedas disponíveis.

Ou seja, consideramos que usamos a moeda para dar o troco, e por isso subtraimos seu valor do troco a ser dado, ou consideramos que não vamos usar a moeda para compor o troco, por isso mantemos o valor do troco e removemos ela como sendo uma possibilidade. Na Figura 1 conseguimos visualizar esse processo com um exemplo geral.

Conseguimos, a partir da explicação acima, perceber que resolvemos o mesmo problema inúmeras vezes; quebramos o problema inicial em subproblemas cada vez menores e mais fáceis de serem resolvidos, cujas soluções usamos para montar a solução geral.

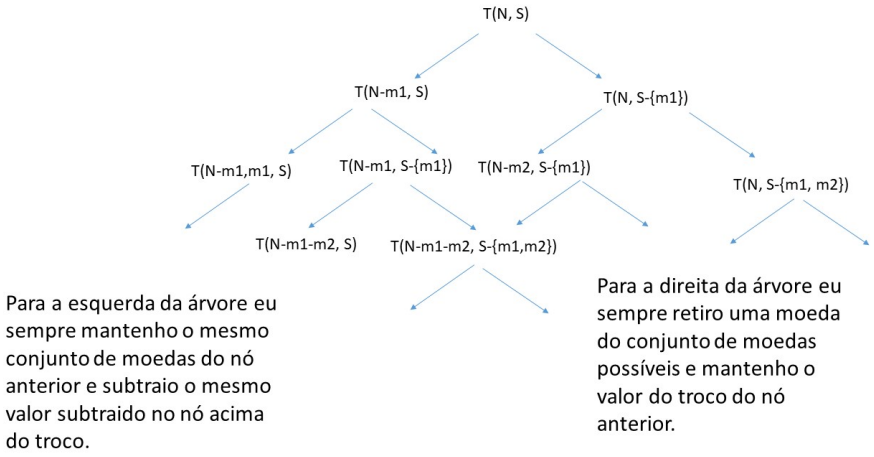

Figura 1: Esquema para montar a árvore de decisões para o problema.

Algo importante ainda a ser discutido é quando parar de montar a árvore. Quando atingimos valores negativos de troco a ser dado, não podemos continuar, pois estaríamos dando dinheiro. Devemos para também quando atingimos troco igual a zero, pois neste caso, fica fácil entender, atingimos uma possível solução para o problema. Finalmente, devemos parar de gerar a árvore quando temos um conjunto vazio de moedas disponíveis, ou seja, não seria possível dar o troco.

3.2.2 Exemplo

Considere que desejamos devolver um troco de valor N = 5, e temos disponíveis moedas de 1, 2 e 3 unidades, T(5, {1, 2, 3}).

Esse exemplo é bem simples e por isso conseguimos identificar facilmente que a solução ideal seria 2. Ou seja, precisamos no mínimo de duas moedas para dar esse troco. E mais, precisamos de uma moeda de 2 e outra de 3. Ao montar o algoritmo entretanto, vamos nos preocupar a somente em obter a resposta de quantas moedas são necessárias.

Montando a árvore do problema temos, por nível da árvore:

- T(4, {1, 2, 3}) e T(5, {2, 3})
- T(3, {1, 2, 3}), T(4, {2, 3}), T(3, {2, 3}) e T(5, {3})
- ...

Conseguimos perceber que mesmo para esse problema bem pequeno e simples a árvore de decisões é consideravelmente grande, como podemos ver na Figura 2. Mas conseguimos montá-la de forma bem simples e com problemas muito fáceis de serem resolvidos.

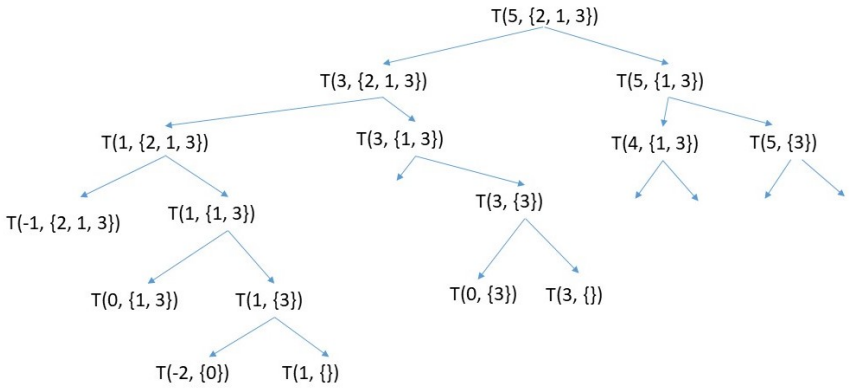

Figura 2: Árvore de decisões do exemplo no qual queremos dar um troco de 5 unidades e temos disponíveis moedas de 1, 2 e 3 unidades.

4 O algoritmo

Considere o seguinte código em python3:
#N é o valor do troco a ser devolvido e S é um vetor com as moedas disponíveis

```
def ProbTrocoMin(N, S):
    Opt = [0 for i in range(0, N+1)]
    numMoedasDisp = len(S)
    for i in range(1, N+1):
      smallest = float("inf")
      for j in range (0, numMoedasDisp):
        if (S[j] ≤ i) :
          smallest = min(smallest, Opt[i − S[j]])
      Opt[i] = 1 + smallest
  returnOpt[N]
```

Nesse código, primeiro definimos um vetor preenchido com zeros que tem dimensão igual ao valor do troco. A seguir, criamos uma variável para armazenar a cardinalidade do conjunto de tipos de moedas que temos disponíveis.

Tendo isso definido, no primeiro *loop*, no qual iteramos de 1 até o valor do troco a ser devolvido, definimos a variável "smallest", a qual setamos para um valor muito grande.

Agora no segundo *loop*, no qual iteramos de 0 até o valor que representa a cardinalidade do conjunto de moedas disponíveis temos uma condição do tipo *if*, onde verificamos se a moeda que estamos analisando é menor que o valor da variável i do primeiro *loop* naquele instante. Caso essa condição seja verdadeira então realizamos a ação ali definida. Note que $S[j] \leq i$, ou seja, nunca teremos valores negativos nessa operação, o que faz todo sentido, pois a usamos para conferir uma posição em um vetor.

Ou seja, estamos quebrando o problema em problemas menores. Quebramos o troco em valores menores, diferenciando sempre em uma unidade, e então analisamos se temos moedas que são menores que esses valores, e portanto poderiam ser utilizadas para dar o troco.

5 Conclusão

Neste trabalho vimos os conceitos básicos e fundamentais de programação dinâmica. Analisando esses conceitos e os exemplos dados conseguimos perceber como ela é uma técnica poderosa para a resolução de problemas que tem uma estrutura específica.

Quanto ao problema do troco, discutimos sobre sua forma geral, como resolvê-lo e uma forma de visualizá-lo que também pode ser aplicada para diferentes problemas para nos ajudar a entendê-los melhor. Além de formularmos matematicamente uma de suas possíveis variações.

Referências

[1] Arenales, M., Morabito, R., Yanasse, H. H., Armentano, V. A., *Pesquisa Operacional*, Elsevier Editora Ltda., Rio de Janeiro, Brasil (2011)

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., *Introduction to Algorithms*, 3th ed., MIT Press, Cambridge, Massachusetts, USA, (2009)

[3] Dasgupta, S., Papadimitriou, C., H., Vazirani, U., *Algorithms*, 1st ed., McGraw-Hill Higher Education, (2006)

[4] Wright, J., W., *The Change-Making Problem*, Journal of the Association for Computing Machinery(1975), [doi:10.1145/321864.321874]