PROBLEMA DA MOCHILA BINÁRIA E OTIMIZAÇÃO COMBINATÓRIA

Eric da Silva Batista
Matemática Aplicada e Computacional
Universidade Estadual do Centro Oeste
Guarapuava, PR
klose.eric31@gmail.com

21 de agosto de 2020

1 Introdução: Otimização Combinatória

Otimização combinatória é um ramo da ciência da computação e da matemática aplicada que estuda problemas de otimização em conjuntos finitos.

Problemas de Otimização combinatória, na sua forma geral, têm como objetivo maximizar ou minimizar uma função definida sobre um certo domínio. A teoria clássica de otimização trata o caso em que o domínio é infinito. Já no caso dos chamados problemas de otimização combinatória, o domínio é tipicamente finito, além disso, em geral é fácil listar os seus elementos e também testar se um dado elemento pertence a esse domínio. Ainda assim, a ideia ingênua de testar todos os elementos deste domínio na busca pelo melhor mostra-se inviável na prática, mesmo para instâncias de tamanho moderado.

Como exemplos clássicos de problemas de otimização combinatória podemos citar o problema do caixeiro viajante, o problema da mochila binária, o problema da cobertura mínima por conjuntos, o problema da floresta de Steiner e o problema da satisfatibilidade máxima. Todos surgem naturalmente em aplicações práticas, tais como o projeto de redes de telecomunicação e de circuitos VLSI, o empacotamento de objetos em containers, a localização de centros distribuidores, o escalonamento e roteamento de veículos, etc. Outras áreas de aplicação incluem a estatística (análise de dados), a economia (matrizes de entrada/saída), a física (estados de energia mínima), a biologia molecular (alinhamento de DNA e proteínas, inferência de padrões), etc.

Considere o seguinte problema. Suponha que uma empresa, chamada *BUBBLE*, é dona de uma máquina de busca e permite que usuário, gratuitamente, entrem com uma ou mais palavras e recebam endereços da web associados às palavras. Apesar do serviço gratuito, a *BUBBLE* lucra apresentando propagandas de outras empresas que pagam para aparecer nas buscas relacionadas a elas. Para isto, a página de busca dispõe de uma região retangular na lateral da tela de busca. Tal região é estreita, de largura fixa, e preenche toda a altura da tela. Todas as propagandas devem ter a mesma largura, porém ter alturas distintas, contanto que não passem da altura da região. Suponha que a busca realizada por um usuário define um conjunto de n propagandas $P = \{1, \ldots, n\}$, associadas à sua busca. Cada propaganda $i \in P$, usa uma certa altura p_i da região e caso seja apresentada ao usuário, fornece um lucro de v_i reais para a *BUBBLE*. Sabendo que a altura da região é B, quais propagandas devem ser apresentadas de maneira a maximizar o valor recebido pela *BUBBLE*?

Este é um típico problema de otimização combinatória. Os possíveis candidatos a solução são os subconjuntos das propagandas, mas são viáveis apenas aquelas que podem ser colocadas na região, sem ultrapassar sua altura. Naturalmente, procura-se uma que maximize o valor arrecadado.

2 Problema da Mochila Binária (PMB)

Dados conjunto de itens $\{1, \ldots, n\}$, cada item i com peso $p_i \geq 0$ e valor $v_i \geq 0$, ambos inteiros, e um inteiro B, encontrar um conjunto $S \subset \{1, \ldots, n\}$ tal que $\sum_{i \in S} p_i \leq B$ e $\sum_{i \in S} v_i$ é o máximo.

De maneira geral, uma entrada I para um problema de otimização combinatória II, deve ter um conjunto de possíveis soluções S_I e uma função v, tal que $v(S)$ é o valor da solução S, para cada $S \in S_I$. O objetivo é encontrar uma solução $S* \in S_I$ tal que $(S*)$ é máximo, se II é de maximização, ou mínimo, se II é de minimização. Neste caso, $S*$ é chamado *solução ótima* de I. Os elementos de S_I são chamados de *soluções viáveis*. Qualquer elemento de S_I poderia ser solução ótima de I, não o sendo apenas por existe outro elemento de S_I com valor melhor para a função objetivo. Por vezes, define-se também um conjunto U_I contendo os candidatos a soluções, com $S_I \subset U_I$, mas não necessariamente um elemento $U \in U_I$ é uma solução viável de I.

No exemplo da mochila binária, o conjunto U_I poderia ser o conjunto de todos os subconjuntos de itens possíveis. Neste caso, nem todos os subconjuntos são possíveis de ser solução, uma vez que a soma dos pesos pode ser maior que a capacidade B da mochila. Os conjuntos em U_I cuja a soma dos pesos é no máximo B são as soluções viáveis de S_I. Dentre estas, deve haver pelo menos uma solução que é ótima.

3 Técnica para solução: Algoritmos de Força-Bruta

Os algoritmos por força-bruta, são algoritmos que, como o nome diz, utilizam de muito esforço computacional para encontrar uma solução, sem se preocupar em explorar as estruturas combinatórias do problema. De fato,estes algoritmos enumeram todas as possíveis solução, verificando sua viabilidade e, caso satisfaça, seu potencial para ser uma solução ótima. Estão entre os algoritmos mais simples, porém, podem demandar grande quantidade de recursos computacionais sendo, no geral, consideradas apenas para entradas de pequeno porte.

Considerando o Problema da Mochila Binária. Qualquer subconjunto de elementos é um potencial candidato a solução do problema, bastando para isso que seu peso total não ultrapasse a capacidade da mochila. Assim, um algoritmo simples, poderia percorrer todos os subconjuntos possíveis e devolver, dentre aqueles que representam soluções viáveis, um de maior valor.

O algoritmo seguinte gera um solução ótima para o problema da mochila. As soluções são representadas por vetores binários, onde a i-ésima posição tem valor 1 se o item i pertence à solução e 0 no caso contrário. O algoritmo é recursivo e além da entrada (B, p, v, n), recebe outros dois parâmetros: $x*$ e x. O parâmetro $x*$ é um vetor binário n-dimensional representa a melhor solução encontrada até o momento, e com isso, todas as referências a $x*$ são para a mesma variável. O parâmetro x é um vetor binário que a cada chamada recursiva, é acrescido de uma nova posição, que representa a pertinência de um item na solução. A primeira chamada do algoritmo é feita com os parâmetros $(B, p, v, n, x*, [])$, onde $x*$ armazena a melhor solução obtida a cada momento, começando com o vetor $\mathbf{0}$, que representa uma solução sem itens. O último parâmetro indica um vetor sem elementos. A base do algoritmo recursivo ocorre quando x contém n elementos binários, momento que representa um subconjunto dos itens da entrada.

Algorithm 1: Mochila-FP(B, p, v, n, x*, x)

Result: Atualiza $x*$ se encontrar solução que completa x, e tem valor melhor
if $|x| = n$ **then**
  if $v \cdot x \leq B$ *e* $v \cdot x > v \cdot x*$ **then**
    $x* \leftarrow x$
else
  Mochila-FB(B, p, v, x*, x||(0));
  Mochila-FB(B, p, v, x*, x||(1));
end

No início de cada chamada os elementos que já tiveram sua pertinência na atual solução definida, estão representados em x, faltando considerar os próximos $n - |x|$ itens. A execução deste algoritmo pode ser representada por uma árvore binária de enumeração, onde cada nó representa uma chamada recursiva e as arestas que ligam seus filhos, representam as duas situações possíveis, quando o item pertence à solução (ramo com $x_i = 1$) ou não (ramo com $x_i = 0$). Note que este algoritmo sempre gera uma árvore de enumeração completa.

No início de cada chamada recursiva, o Algoritmo Mochila-FP precisa considerar a pertinência dos próximos $n - |x|$ itens, sendo que x é acrescido de uma coordenada a cada chamada recursiva. O algoritmo pode ser implementado de maneira a ter complexidade de tempo limitada pela recorrência $T(n) = 2T(n-1) + O(1)$, cuja resolução mostra que $T(n)$ é $O(2^n)$. Note que o primeiro *if* pode ser implementado em tempo constante, mantendo para cada vetor que representa uma solução, uma variável contendo o valor total dos itens atribuídos. Tal passo verificará todos os subconjuntos possíveis de itens, levando a um algoritmo bastante custoso na prática.