PROJETO E ANÁLISE DE ALGORITMOS (INF 2926)

Departamento de Informática, PUC-Rio
Prof. Thibaut Vidal

André Luiz de Brandão Damasceno (1621797)
Tássio Ferenzini Martins Sirqueira (1622006)

1. Problema da Mochila

O problema da mochila requer que um subconjunto de itens sejam escolhidos, de tal forma que o somatório de seus valores seja maximizado sem exceder a capacidade da mochila. O problema pode ser enunciado da seguinte forma:

Dados um número m (tamanho da mochila) ≥ 0, para cada item i variando de {1, ..., n}, um número v_i (valor do item) ≥ 0 e um número p_i (peso do item) ≥ 0, encontrar um subconjunto S de {1, ..., n} que maximize v(S) sob a restrição p(S) ≤ m. Onde, v(S) denota a soma $\sum_{i \in S} v_i$ e, analogamente, p(S) denota a soma $\sum_{i \in S} p_i$.

O objetivo do problema é então encontrar uma coleção de objetos, a mais valiosa possível, que respeite a capacidade da mochila. Todavia os problema da Mochila pertencem a família NP-Difícil. Neste trabalho todos os algoritmos apresentados resolvem o Problema da Mochila, que consiste em escolher n itens, tais que o somatório dos valores é maximizado sem que o somatório dos pesos ultrapassem a capacidade da Mochila. Isto pode ser formulado como o seguinte problema de maximizar:

$$\text{Maximizar } \sum_{k=0}^{n} v_k x_k$$

$$\text{Sujeito } \sum_{k=0}^{n} p_k x_k \leq M$$

$$\text{Sendo } x_k \in \{0, 1\}, k = 1 \ldots n$$

onde x_k é uma variável binária igual a 1 se k deve ser incluído na mochila e 0 caso contrário, considerando o problema da mochila inteira.

Existem diversas variações do problema da mochila, sendo os mais comuns o problema da Mochila 0/1 (Binária), onde cada item pode ser escolhido no máximo uma vez, a mochila fracionária, onde os itens podem ser divididos para entrar na mochila, o problema da Mochila Limitado, onde temos uma quantidade limitada para cada tipo de item. Já o problema da Mochila com Múltipla Escolha ocorre quando os itens devem ser escolhidos de classes disjuntas, e se várias Mochilas são preenchidas simultaneamente temos o problema da Mochila Múltipla. A forma mais geral é o problema da Mochila com múltiplas restrições é basicamente um problema de Programação Inteira Geral com Coeficientes Positivos. Neste trabalho serão abordados a Mochila Fracionária, a Mochila Binária e a Mochila Binária com Conflitos, utilizando Método Guloso e Programação Dinâmica.

2. Solução usando o Método Guloso para Mochila Fracionário

O método Guloso sugere que podemos construir uma algoritmo que trabalhe considerando uma entrada por vez. A cada etapa, uma decisão é tomada considerando se uma entrada particular é uma solução ótima. Isto é realizado considerando as entradas numa ordem determinada por algum processo de seleção.

No problema da mochila fracionária temos n objetos para serem colocados em uma mochila de capacidade m. Cada objeto i tem peso p_i e valor v_i. Posso escolher uma fração (entre 0% e 100%) de cada objeto para colocar na mochila. Considerando p_n como o vetor de pesos, v_n o vetor de valores. Temos que uma mochila fracionária viável é qualquer vetor x_n tal que $x * p \leq m$ e $0 \leq x_i \leq 1$ para todo i. O valor de uma mochila x é o número x * v. O problema é encontrar uma mochila viável de valor máximo.

Suponha $m = 50$ e $n = 3$ e os valores com os pesos da Tabela 1. Utilizando a solução do problema da mochila fracionária, o valor dessa solução é x * v = 1040. (A solução correspondente no problema inteiro tem valor 1000, mesmo na melhor heurística.)

Tabela 1 –Solução da mochila fracionária.

N	1	2	3
P	40	30	20
V	840	600	400
V/P	21	20	20
P/V	0,047	0,05	0,05
X	1	0,333...	0

O seguinte algoritmo guloso resolve o problema da mochila fracionária.

```
Mochila-Fracionária (p, v, n, m) {

        j ← n

        enquanto j ≥ 1 e p_j ≤ m faça
                x_j ← 1
                m ← m − pj
                j ← j − 1

        se j ≥ 1 então
                xj ← v_j/p_j

                para i ← j−1 decrescendo até 1 faça
                        xi ← 0
    devolva x

}
```

O consumo de tempo do algoritmo é O(n), não considerando o tempo Θ (n log n) (HEAPSORT) necessário para colocar os objetos em ordem decrescente utilizando a razão valor/peso.

3. Solução usando o Método Guloso para Mochila Binária

O método Guloso é a técnica de projeto mais simples que pode ser aplicada a uma grande variedade de problemas. A grande maioria destes problemas possui n entradas e é requerido obter um subconjunto que satisfaça alguma restrição. Qualquer subconjunto que satisfaça esta restrição é chamado de solução viável. Uma solução viável que satisfaça a função objetivo é chamada de solução ótima. Existem várias maneiras de determinar uma solução viável, mas não necessariamente uma solução ótima. Se a inclusão da próxima entrada na solução ótima construída parcialmente resultará numa solução inviável, então esta entrada não será adicionada à solução parcial. O processo de seleção em si é baseada em alguma medida de otimização. O algoritmo escolhido para esta implementação usa a estratégias gulosas onde em cada passo será incluído um objeto que tem uma das heurística apresentadas a seguir. Isto significa que o objeto será considerado na ordem definida pela heurística que foi aplicada, resolvendo o problema da Mochila usando a estratégia gulosa.

Esse algoritmo realiza os seguintes passos:

```
Mochila-Gulosa (p, v, m, n) {

X ← 0
total ← m

Para i ← 1 até n faça
        Se pi > total então Exit;
        Xi ← 1
        total ← total − p_i

 Para j ← i até n faça
        Se p_j <= total então
                X(i) ← 1
                total ← total − p_j

}
```

O método guloso gasta tempo de O(n) no pior caso. Contudo, o método guloso não garante que seja atingido o valor ótimo, sendo na maioria dos casos apenas uma solução viável. Um exemplo disto pode ser visto na Tabela 1, onde utilizando o método guloso e colocando na mochila os itens na ordem que são lidos. Desta maneira o item 1 entra na mochila com o valor de 840, e os demais itens não entram pois o peso ultrapassa a capacidade da mochila, entretanto, a melhor solução é colocar na mochila os itens 2 e 3 que atingiria o valor de 1000 e o máximo de peso da mochila.

4. Solução usando Programação Dinâmica para Mochila Binária

Programação dinâmica é uma técnica que tem como objetivo tratar o número de combinações geradas pela solução. A ideia é armazenar em uma tabela as soluções das instâncias do problema, onde podemos definir que para i = {0, …, n} e b = {0, …, m}, t[i, b] é o valor de uma solução da instância p_i, v_i, b do problema. A tabela deve satisfazer as seguinte recorrência: para todo $i \geq 1$ e todo b, $t[i,b]=t[i-1,b]$ se $p_i > b$ e $\max (t[i-1,b] , t[i-1,b-p_i] + v_i)$ se $p_i \leq b$.

O algoritmo abaixo resolve o problema da mochila 0/1 utilizando programação dinâmica.

```
Mochila-Prog-Din (p, v, n, m) {

para b ← 0 até m faça
        t[0,b] ← 0

        para i ← 1 até n faça
          a ← t[i−1,b]

          se p_i > b então
                a′ ← 0
          senão
                a′ ← t[i−1,b−p_i] + v_i

          t[i,b] ← max(a, a′)

b ← m
X ← {}

para i ← n decrescendo até 1 faça

     se t[i,b] ≠ t[i−1,b] então
          X ← X ∪ {i}
          b ← b − pi

devolva X

}
```

O consumo de tempo do algoritmo é Θ (n * m) para todos os casos. Considerando *m* = 5 e *n* = 3 (Tabela 2), a solução pelo método dinâmico é 22, conforme Tabela 3.

Tabela 2 –Itens disponíveis.

N	1	2	3
P	1	2	3
V	6	10	12
V/P	6	5	4

Tabela 3 –Solução pelo método dinâmico.

/	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	6	10	16	16	16
3	0	6	10	12	12	**22**

Fazendo uma comparação entre os algoritmos, o método guloso pode chegar em uma solução ótima, mas depende da heurística aplicada e da entrada dos dados, podendo esta solução ótima não ser atingida. Já o método dinâmico sempre chega a uma solução ótima, entretanto a divergência está no tempo de execução dos algoritmos. O método guloso leva tempo O (n) + O (n log n) do HeapSort dependendo da heurística aplicada para a resolução, enquanto o método dinâmico gasta tempo Θ (n * m).

5. Solução usando uma Heurística Gulosa para Mochila Binária com Conflitos

A heurística gulosa é iniciada ordenando os itens utilizando o Heapsort de duas formas diferentes: considerando os valores e considerando a razão (valor/peso). Após a ordenação ser realizada, os itens são inseridos na mochila, levando em consideração o peso total da mochila e o grafo de conflitos, para que não seja colocada na mochila itens que não possam ficar juntos. Para isso foi utilizado um vetor de incidência de conflitos que era iniciado com valor “false” em todos os índices. Os índices do vetor corresponde ao identificador (id) de cada item. Assim, a medida em que um item é inserido na mochila, é verificado quais os itens possuem conflito com ele. Em seguida, o item com conflito é marcado como “true” no valor correspondente ao seu índice no vetor de incidência. Vale ressaltar que são geradas uma mochila para cada ordenação. Conforme podemos observar no pseudocódigo a seguir, o resultado final é a mochila que gerou o maior lucro.

```
Mochila-Heuristica-Gulosa (itens, pesoMaxMochila) {
        itensOrdenado1 ← HeapSortValor(itens)
        itensOrdenado2 ← HeapSortRazao(itens)

        mochila1 ← InserirMochila(itensOrdenado1, pesoMaxMochila)
        mochila2 ← InserirMochila(itensOrdenado2, pesoMaxMochila)

        retorna maxValor(mochila1, mochila2)
}
```

```
InserirMochila (itens, pesoMochila) {
        total ← pesoMochila
        incidenciaConflitos [ |itens| ]

        Para i ← 1 até |itens| faça
                incidenciaConflitos  [ id(i) ] ← false

        Para i ← 1 até |itens| faça
                Se peso(i) > total & incidenciaConflitos [ id(i) ] = true então continue

                mochila ← i
                total ← total - peso(i)

                Se total = 0 então break

                vetorConflitos[] ← conflitos(i)

                Para i ← 1 até | vetorConflitos[] | faça
                        id = vetorConflitos[i]
                        incidenciaConflitos  [id] ← true

        retorna mochila
}
```

A solução para o problema da Mochila Binária com Conflitos é composta por dois procedimentos de ordenação utilizando o HeapSort com complexidade O(n logn), um vetor de conflito com complexidade de O(n), e o Método Guloso para inserção dos itens na mochila a complexidade de O(n * m), onde n é o número de itens e m é o número de conflitos. Logo, podemos concluir que a complexidade geral do algoritmo é O(n * m).

6. Referências

CALDAS, Ruiter Braga. Projeto e Análise de Algoritmos. Universidade Federal de Minas Gerais, 2004.

CORMEN, Thomas H. Introduction to algorithms. MIT press, 2009.

FEOFILOFF, Paulo. Análise de algoritmos. Internet: http://www.ime.usp.br/~pf/analise_de_algoritmos, v. 2009, 1999.

KLEINBERG, Jon; TARDOS, Eva. Algorithm design. Pearson Education India, 2006.

ZIVIANI, Nivio et al. Projeto de algoritmos: com implementações em Pascal e C. Thomson, 2004.