Universidade Federal de Minas Gerais

Instituto de Ciências Exatas

Departamento de Ciência da Computação

PROJETO E ANÁLISE DE ALGORITMOS

Trabalho: disponível em: http://www.dcc.ufmg.br/~ruiter/paatp2

Ruiter Braga Caldas

Professor - Nivio Ziviani

Belo Horizonte
10 de maio de 2004

Sumário

1 O Problema da Mochila

O problema da Mochila (*knapsack problem*) pode ser enunciado da seguinte forma: Dados um número $m \geq 0$, um inteiro positivo n e, para cada i em $\{1, \ldots, n\}$, um número $v_i \geq 0$ e um número $w_i \geq 0$, encontrar um subconjunto S de $\{1, \ldots, n\}$ que maximize $v(S)$ sob a restrição $w(S) \leq m$. Onde, $v(S)$ denota a soma $\sum_{i \in S} v_i$ e, analogamente, $w(S)$ denota a soma $\sum_{i \in S} w_i$.
Os números v_i e w_i podem ser interpretados como utilidade e peso respectivamente de um objeto i. O número m pode ser interpretado como a capacidade de uma mochila, ou seja, o peso máximo que a mochila comporta. O objetivo do problema é então encontrar uma coleção de objetos, a mais valiosa possível, que respeite a capacidade da mochila.
Este problema vem sendo estudado deste o trabalho de D.G. Dantzig[5], devido a sua utilização imediata na Indústria e na Gerencia Financeira, porém foi mais enunciado por razões teóricas, uma vez que este freqüentemente ocorre pela relaxação de vários problemas de programação inteira. Toda a família de **Problemas da Mochila** requer que um subconjunto de ítens sejam escolhidos, de tal forma que o somatório das suas utilidades seja maximizado sem exceder a capacidade da mochila. Diferentes tipos de problemas da Mochila ocorrem dependendo da distribuição de ítens e Mochilas como citado em [5]:
No problema da Mochila 0/1 (0/1 *Knapsack Problem*), cada ítem pode ser escolhido no máximo uma vez, enquanto que no problema da Mochila Limitado (*Bounded Knapsack Problem*) temos uma quantidade limitada para cada tipo de ítem. O problema da Mochila com Múltipla Escolha (*Multiple-choice Knapsack Problem*) ocorre quando os ítens devem ser escolhidos de classes disjuntas, e se várias Mochilas são preenchidas simultaneamente temos o problema da Mochila Múltiplo (*Multiple Knapsack Problem*). A forma mais geral é o problema da Mochila com multi-restrições (*Multi-constrained Knapsack Problem*) o qual é basicamente um problema de Programação Inteira Geral com Coeficientes Positivos.
Todos os problemas da Mochila pertencem a família *NP-Hard*[5], significando que é muito improvável que possamos desenvolver algoritmos polinomiais para este problema. Porém, a despeito do tempo para o pior caso de todos os algoritmos terem tempo exponencial, diversos exemplos de grandes instâncias podem ser resolvidos de maneira ótima em fração de segundos. Estes resultados surpreendentes vem de várias décadas de pesquisa que tem exposto as propriedades estruturais especiais do Problema da Mochila, que tornam o problema tão relativamente fácil de resolver.
Neste trabalho todos os algoritmos apresentados resolvem o *Problema da Mochila 0/1*, que consiste em escolher n ítens, tais que o somatório das utilidades é maximizado sem que o somatório dos pesos extrapolem a capacidade da Mochila. Isto pode ser formulado como o seguinte problema de maximizar :

$$Maximizar \sum_{j=1}^{n} u_j x_j$$

$$Sujeito \sum_{j=1}^{n} p_j x_j \leq M$$

$$x_j \in \{0, 1\}, j = 1 \ldots n$$

onde x_j é uma variável binária igual a 1 se j deve ser incluído na mochila e 0 caso contrário.

1.1 Provar que o Problema da Mochila é NP-Completo

Para provar que um problema $\prod$ pertence a NP-Completo é necessário provar, de acordo com [6], que:

- o mesmo pertence a classe NP, apresentando em algoritmo não-determinista em tempo polinomial ou mostrando que uma dada solução pode ser verificada em tempo polinomial.
- Apresentar um redução em tempo polinomial de um problema NP-completo para o mesmo.

1.2 Mostrar que o Problema está em NP

Para provar que o *Problema da Mochila 0/1* pertence a classe NP vamos apresentar um algoritmo não-determinista que resolva o problema em tempo polinomial.

KNAPSACKND$(cM, uM, n, P[1..n], U[1..n], X[1..n])$
1 **for** $i \leftarrow 1ton$
2 **do**
3 $\quad X[i] \leftarrow escolhe(0, 1);$
4 **if** $(\sum_1^n P[i] * X[i] > cM) OR (\sum_1^n U[i] * X[i] < uM)$
5 $\quad$ **then return** $Insucesso$
6 $\quad$ **else return** $Sucesso$

O procedimento $KnapsackND$ é uma algoritmo Não-Determinista para o problema de decisão da Mochila. As linhas de 1-3 atribui o valor 0/1 para o vetor solução $X[i], 0 \leq i \leq n$. Na linha 4 é feito um teste para verificar se a atribuição do pesos é viável e não ultrapassa a capacidade da Mochila cM e se o resultado da Utilidade é pelo menos uM. Uma solução com sucesso é encontrada se as restrições são satisfeitas. A complexidade de tempo deste procedimento é $O(n)$. Se m é o tamanho da entrada usando uma representação binária, o tempo é $O(m)$[4].

1.3 Redução Polinomial

Vamos apresentar uma redução polinomial a partir de outro problema conhecido como *NP*-Completo para o problema da Mochila.
Antes de fazê-lo vamos dar uma versão de um problema de decisão para o PROBLEMA DA MOCHILA 0/1. O problema de Otimização difere do problema de Decisão somente na função de Otimização. Assim a versão de Decisão para o problema da Mochila é a seguinte:
Entrada: um conjunto de n ítens, tal que todo i tem utilidade c_i e peso w_i, uma mochila tem capacidade W e um valor positivo C.
Questão: Existe um subconjunto $S \subseteq \{1, ..., n\}$ de ítens, tais que o peso total é no máximo W:

$$\sum_{i \in S} w_i \leq W$$

e o valor da utilidade total é pelo menos C:

$$c(S) = \sum_{i \in S} c_i \geq C?$$

Para provar que a versão de decisão do problema da Mochila é NP-completo vamos fazer uma redução polinomial a partir do problema MAXIMUM SUBSET SUM
Solução: A versão do problema de decisão para MAXIMUM SUBSET SUM é definida como apresentada em [3]:
Entrada: Um conjunto finito A, um tamanho inteiro positivo s_i para cada elemento $i \in A$, e um inteiro positivo B.
Questão: Existe uma subconjunto $A' \subseteq A$ tal que

$$\sum_{i \in A'} s_i = B?$$

Como podemos notar, MAXIMUM SUBSET SUM é um caso especial do problema de Otimização do PROBLEMA DA MOCHILA 0/1 com $c_i = w_i$, como citado em [2].
A redução completa é como se apresenta a seguir:
Dados uma instância do problema SUBSET SUM, reduziremos esta para uma instância do PROBLEMA DA MOCHILA 0/1 da seguinte forma :

$$U = A, \; w_i = c_i = s_i, \; W = C = B$$

Esta redução é feita em tempo polinomial, desde que todas as atribuições são executadas em tempo polinomial. Desta forma uma, resposta para uma instância do problema da Mochila corresponde a uma resposta para o problema MAXIMUM SUBSET SUM.

- Uma resposta "sim"para uma instância do PROBLEMA DA MOCHILA 0/1 significa que existe um subconjunto $S' \in U$ tal que

  $$\sum_{i \in S'} w_i \leq W \text{ and } \sum_{i \in S'} c_i \geq C$$

  Isto significa, usando a nossa transformação, que existe um subconjunto $A' \in A$ tal que

  $$B \leq \sum_{i \in A'} s_i \leq B.$$

  Isto é,

  $$\sum_{i \in A'} s_i = B.$$

  Assim, por definição, é uma resposta "sim"para problema SUBSET SUM.

- Um resposta "Não"para o PROBLEMA DA MOCHILA 0/1, significa que tal conjunto não existe. O que é, exatamente, uma resposta negativa para o problema SUBSET SUM.

Assim mostramos que:

- o problema da Mochila está em NP e,
- existe uma redução polinomial a partir do problema MAXIMUM SUBSET SUM para o PROBLEMA DA MOCHILA 0/1.

Com isso provamos que PROBLEMA DA MOCHILA 0/1 é NP-Completo.

2 Solução usando Backtracking

Backtracking é uma estratégia para sistematicamente examinar a lista de possíveis soluções. A idéia do backtracking é eliminar a verificação explícita de uma boa parte dos possíveis candidatos. Para tanto, o problema deve respeitar as restrições que maximizam/minimizam alguma função de otimização. Os seguintes passos são respeitados:

1. Definir um espaço de solução para o problema. Este espaço de solução deve incluir pelo menos uma solução ótima para o problema.

2. Organizar o espaço de solução de forma que seja facilmente pesquisado. A organização típica é uma árvore.

3. Proceder a busca em profundidade.

Backtracking é uma estratégia que se aplica em problemas cuja solução pode ser definida a partir de uma seqüência de n decisões, que podem ser modeladas por uma árvore que representa todas as possíveis seqüências de decisão. De fato, se existir mais de uma disponível para cada uma das n decisões, a busca exaustiva da árvore é exponencial. A eficiência desta estratégia depende da possibilidade de limitar a busca, ou seja, podar a árvore, eliminando as sub-árvores que não levam a nenhuma solução. As soluções são representadas por n-tuplas ou vetores de solução $(v_1, v_2, \ldots, v_n)$. Cada v_i é escolhido a partir de um conjunto finito de opções S_i. O algoritmo inicia com um vetor vazio e em cada etapa o vetor é extendido com um novo valor formando um novo vetor que pode representar uma solução parcial do problema. Na avaliação de um vetor $(v_1, \ldots, v_i)$, se for constatado que ele não pode representar uma solução parcial, o algoritmo faz o backtracking, eliminando o último valor do vetor, e continua tentando extender o vetor com outros valores alternativos. Implementamos o algoritmo descrito em [4], este problema possui espaço de solução consistindo de 2^n maneiras distintas de atribuir zero ou um para o vetor de solução X. Este algoritmo faz uso de uma função limite para ajudar a eliminar alguns nós sem fazer a expansão deles. Uma boa função limite para este problema é obtida usando um limite superior com o valor da melhor solução viável obtida pela expansão de um nó ativo e qualquer dos seus descendentes. Se este limite superior não for maior que valor da melhor solução encontrada até o momento, então este nó pode ser eliminado. A função limite funciona da seguinte forma: Se num nó Z qualquer, o valor de x_i,$1 \leq i \leq k$ já foi determinado, então um limite superior para Z pode ser obtido pela relaxação do requisito de $x_i = 0$ ou $x_i = 1$ por $0 \leq x_i \leq 1$ para os nós $k + 1 \leq i \leq n$ e usando um algoritmo guloso para resolver o problema da relaxação. A função *Limite* determina um limite superior sobre a melhor solução obtida pela expansão de um nó Z no nível $k + 1$ do espaço de estados.

```
LIMITE(ut, pt, k, M)
b ← ut
c ← pt
for i ← k + 1 to n
do
    c ← c + P(i)
    if c < M
       then b ← b + U(i)
       else  return (b + (1 − (c − M)/P(i)) ∗ U(i))
    return b
```

O algoritmo seguinte implementa o método de Backtracking:

```
METODO_BACKTRACKING(M, n, P, I, pf, if, X)
pc ← ic ← 0
k ← 1
if ← −1
while 1 = 1
do
    while (k ≤ n)&(pc + P(k) ≤ M)
    do
        pc ← pc + P(k)
        ic ← ic + I(k)
        Y(k) ← 1
        k ← k + 1
    if k > n
       then if ← ic
            pf ← pc
            k ← n
            X ← Y
       else  Y(k) ← 0
    while Limite(ic, pc, k, M) ≤ if
    do
        while (k ≠ 0)&(Y(k) ≠ 1)
        do
            k ← k − 1
        if k = 0
           then return
        Y(k) ← 0
        pc ← pc − P(k)
        ic ← ic − I(k)
    k ← k + 1
```

O ordem de complexidade de espaço para este algoritmo é $O(2^n)$ de acordo com [4]. Da função Limite segue que o limite para um nó filho à esquerda viável de um nó Z é o mesmo para Z. Então a função Limite não precisa ser usada sempre que o algoritmo faz um movimento para o filho a esquerda de um nó. Desde que o algoritmo tentará fazer um movimento para a esquerda sempre que houver uma escolha entre esquerda e direita, a função Limite será chamada somente depois de uma série de movimento com sucesso para os filhos a esquerda. Quando $if \neq -1$, $X(i), 1 \leq i \leq n$ é tal que $\sum_{i=1}^{n} I(i)X(i) = if$. No *while* nas linhas 6 a 11 movimentos sucessivos são feitos para filhos a esquerda viável. $Y(i) = 1$, $1 \leq i \leq n$ é o caminho para o nó corrente. $pc = \sum_{i=1}^{k-1} P(i)Y(i)$ e $ic = \sum_{i=1}^{k-1} I(i)Y(i)$. Se na linha 12, $k > n$ então $ic > if$ indicando que o caminho para esta folha terminou na última vez que a função Limite foi usada. Se $k \leq n$ então $P(i)$ não cabe e um movimento para um filho à direita deve ser feito. Então $Y(k)$ é marcado como 0 na linha 17. Se na linha 18, $Limite \leq n$, então o caminho corrente terminou e ele não pode levar a uma solução melhor que a encontrada até o momento. Nas linhas 20 a 22, retornamos ao longo do caminho para o nó mais recente a partir do qual um movimento não tentado pode ser feito. Se não existe este caminho então o algoritmo termina na linha 23-24. Caso contrário $Y(k), pc$ e ic são apropriadamente atualizados para corresponder a um movimento para a direita. O limite para este novo nó é calculado. O processo de retorno das linhas 18 a 27 continua até que um movimento é feito para um filho a direita a partir do qual exista uma possibilidade de obter uma solução com valor maior que if. Considerando o seguinte exemplo para o problema da Mochila: A

n	1	2	3	4	5	6	7	8
U	11	21	31	33	43	53	55	65
P	1	11	21	23	33	43	45	55

Figura 1: Exemplo para o Backtracking

árvore mostra as várias escolhas que são feitas para o vetor de solução parcial Y. O i-ésimo nível da árvore corresponde a uma atribuição 0 ou 1 para $Y(i)$, quando inclui ou exclui o Peso $P(i)$. Os dois números no nó são o peso corrente (pc) e a importância corrente (ic). O nós que não contem números tem peso e utilidade idêntico aos pais. O número de fora do nó a direita é o limite do nó. O limite dos nós a esquerda são os mesmos dos pais. A variável if do algoritmo é atualizada em cada nó A, B, C e D. Cada vez que if é atualizada, o vetor solução final X também é atualizado. Ao terminar $if = 159$ e X=(1,1,1,0,1,1,0,0). Dos $2^9 - 1 = 511$ nós do espaço de estados da árvore somente 33 são gerados.

3 Solução usando Programação Dinâmica

Uma solução ótima baseada no método guloso é definida por uma seqüência de decisões locais ótimas, como pode ser visto na próxima seção. Quando o método guloso não funciona, uma possível saída seria a geração de todas as possíveis seqüências de decisões. E a melhor seqüência seria então escolhida, como no caso do Backtracking acima. Essa solução é de ordem exponencial e, portanto, ineficiente. Programação dinâmica é uma técnica que tem como objetivo diminuir o número de seqüências geradas. A programação dinâmica trata o número de combinações da seguinte forma: Vamos considerar n ítens, dentre os quais devemos escolher r. Para escolher os r ítens, podemos proceder de duas formas:

1. Escolher o primeiro ítem. Escolher depois $r-1$ ítens dos $n-1$ ítens restantes.

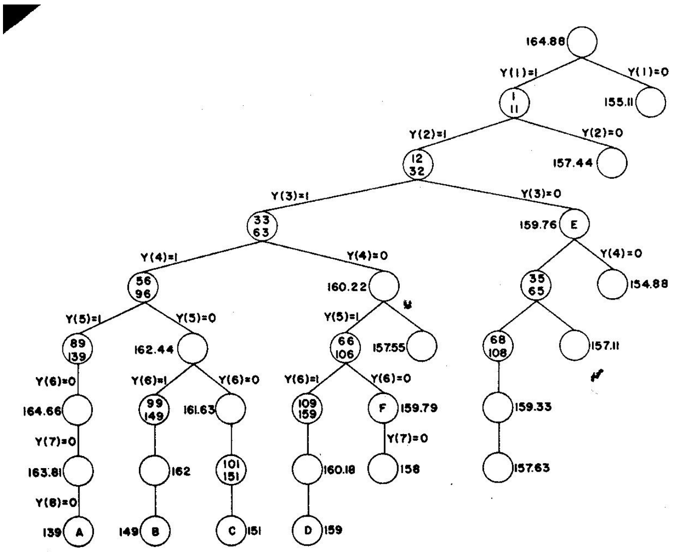

Figura 2: Árvore gerada pelo Algoritmo Guloso

2. Não escolher o primeiro ítem. Então devemos escolher r ítens dos $n-1$ ítens restantes

Está solução pode ser traduzida da seguinte forma: $\binom{n}{r} = \binom{n-1}{r-1} + \binom{n}{r-1}$ Se usarmos uma estratégia de divisão e conquista para implementar esta solução obteremos um algoritmo com complexidade $O(2^n)$, sendo o maior problema desta abordagem o número de vezes que o mesmo problema é resolvido. Uma outra abordagem seria usar uma tabela para armazenar as soluções que vão se repetir, e é essa a idéia principal da programação dinâmica. Usando esta abordagem para o problema podemos melhor a complexidade deste problema para $O(n^2)$. Sendo o projeto de uma algoritmo baseado em Programação Dinâmica dividido em duas partes:

1. Identificação
   - determinar uma solução por divisão e conquista.
   - Analisar e verificar que o tempo de execução é exponencial.
   - Mesmo sub-problema resolvido várias vezes.
2. Construção
   - Pegar a parte do algoritmo de divisão e conquista que corresponde à parte da conquista e substituir as chamadas recursivas por uma olhada na tabela.
   - Em vez de retornar um valor, armazená-lo na tabela.
   - Usar o caso base do algoritmo de divisão e conquista para inicializar a tabela.
   - Determinar o padrão de preenchimento da tabela.
   - Definir um laço que utiliza o padrão para preencher os demais valores da tabela.

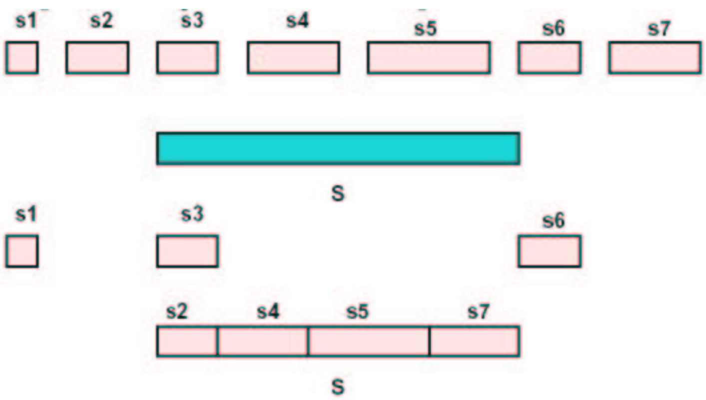

Figura 3: Instância para o Algoritmo de Programação Dinâmica

Vamos apresentar um exemplo do problema da Mochila onde a utilização da programação dinâmica permite encontrar a solução ótima. Sejam n ítens de tamanhos $s1, s2, \ldots, sn$, conforme a Figura 3. A idéia é verificar se existe um subconjunto desses ítens cujo tamanho total seja exatamente S. Numa solução por divisão e conquista, devemos ter problemas menores da mesma natureza. Podemos Generalizar para a situação em que temos i ítens e o tamanho total é j. Para saber se retornamos verdadeiro, temos que analisar duas possibilidades:

1. O i-ésimo ítem é usado para completar o tamanho j.
2. O i-ésimo item não é usado e, portanto j é alcançado até os $i-1$ primeiros ítens.

Devemos usar uma tabela $t[n, S]$ para armazenar t, caso seja possível completar S com os n elementos. Caso não seja possível, preenchemos com f. Pelo definido acima uma célula $t[i, j]$ deve ser preenchida com t se uma das duas situações é verdadeira:$t[i-1, j-s_i]$ ou $T[i-1, j]$. O padrão de preenchimento seria então o mesmo apresentado na Figura 4

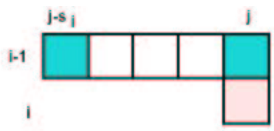

Figura 4: Padrão de preenchimento da tabela

Sendo assim a tabela final teria o formato da Figura 5:
O algoritmo utilizado na implementação foi extraído de:

- http://www.mpi-sb.mpg.de/ rybal/armc-live-termin/node5.html
- http://www-cse.uta.edu/ holder/courses/cse2320/lectures/l15/node12.html
- http://www.cse.uni.edu/ goddard/Courses/CSCE310J

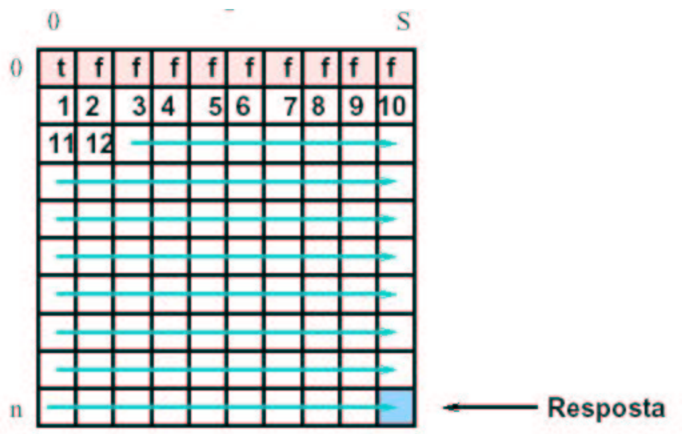

Figura 5: Formato Final da Tabela

```
Metodo_Dinâmico(v[1..n], w[1..n], n, W)
 1  for w ← 0 to W
 2  do
 3      c[0, w] ← 0
 4  for i ← 1 to n
 5  do
 6      c[i, 0] = 0
 7  for i ← 1 to n
 8  do
 9      for w ← 1 to W
10      do
11          if w[i] ≤ w
12            then
13                if v[i] + c[i − 1, w − w[i]] > c[i − 1, w]
14                  then
15                       c[i, w] = v[i] + c[i − 1, w − w[i]]
16                  else c[i, w] = c[i − 1, w]
17            else c[i, w] = c[i − 1, w]
```

Podemos deduzir a complexidade deste algoritmo através da manipulação que é executada em cada laço, sendo que os dois laços iniciais são apenas para inicializar a tabela, sendo o laço da linha 1 a 3 da ordem $O(W)$, onde W é a capacidade da Mochila, e o laço da linha 4 a 6 da ordem $O(n)$, onde n é o número de ítens. Os laços seguintes realizam a construção da tabela, o laço da linha 7 a 17 executa $O(n)$ e o laço da linha 9 a 17 executa $O(W)$ vezes, sendo este algoritmo da ordem de $O(n * W)$. O algoritmo acima encontra apena o maior valor possível que pode ser alocado na Mochila, para saber quais os ítens que tornam este valor máximo outro algoritmo foi implementado:

```
BACKTRACE(n, W)
i ← n
j ← M
while ((i > 0)&(j > 0))
do if (c[i, j] <> c[i − 1, j])
      then X[i] ← 1
           j ← j − w[i]
           i − −
      else  X[i] ← 0
           i − −
```

4 Solução usando o Método Guloso

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. Queremos então encontrar uma solução viável que maximize ou minimize uma dada função objetivo. Uma solução viável que satisfaça a função objetivo é chamada de solução ótima. Existem maneiras óbvias de determinar uma solução viável, mas não necessariamente uma solução ótima[4]. O método Guloso sugere que podemos construir uma algoritmo que trabalhe em estágios, considerando uma entrada por vez. Em cada estágio, uma decisão é tomada considerando se uma entrada particular é uma solução ótima. Isto é conseguido considerando as entradas numa ordem determinada por algum processo de seleção. Se a inclusão da próxima entrada na solução ótima construída parcialmente resultará numa solução inviável, então esta entrada não será adicionada a solução parcial. O processo de seleção em si é baseada em alguma medida de otimização. Esta medida pode ou não ser a função objetivo. Na verdade várias medidas de otimização diferentes podem aplicáveis para um determinado problema. Muitas destas, entretanto, resultarão em algoritmos que gerarão solução sub-ótimas.

O algoritmo escolhido para esta implementação usa a estratégias gulosas mais interessante, de acordo com [4], este estratégia faz uma negociação entre a taxa em que a utilidade decresce e o peso é usado. Em cada passo será incluído um objeto que tem a maior utilidade pro unidade de peso usado. Isto significa que o objeto será considerado na ordem decrescente da Maior Utilidade sobre o Peso $(U(i)/P(i))$. Se os objetos estiverem ordenado numa ordem decrescente de Utilidade sobre o Peso $(U(i)/P(i) \geq U(i+1)/P(i+1))$ o algoritmo abaixo, extraido de [4] e adaptado a partir da linha 9 a 13 para resolver o problema da Mochila 0/1, resolve o problema da Mochila usando a estratégia gulosa. Sem considerar o tempo de ordenação da entrada, que na implementação foi usado o algoritmo do *quicksort* extraído do livro [6], o algoritmo abaixo executa a estratégia gulosa em tempo $O(N)$. Esse algoritmo realiza os seguintes passos:

- o vetor solução X é inicializado;
- a capacidade utilizada da mochila é armazenado em cum;

- O primeiro laço FOR, da linha 3 a 8, colocando cada ítem na mochila, subtraindo o peso do ítem do valor de cum. Isso é feito até que a cum não comporte o próximo peso. Neste ponto todos os ítem de maior relação $U(i)/P(i)$ foram colocado na Mochila.
- O segundo laço FOR, da linha 9 a 13, muda de estratégia e procura nos ítens restantes aquele(s) que possua um peso que caiba no valor restante de cum para preencher o espaço da Mochila.

Metodo_Guloso($U[1..n], P[1..n], M, n$)

1 $X \leftarrow 0$
2 $cum \leftarrow M$
3 **for** $i \leftarrow 1$ **to** n
4 **do**
5   **if** $P(i) > cum$
6   **then** $Exit$;
7     $X(i) \leftarrow 1$;
8     $cum \leftarrow cum - P(i)$
9 **for** $j \leftarrow i$ **to** n
10 **do**
11   **if** $P(j) <= cum$
12   **then** $X(i) \leftarrow 1$;
13             $cum \leftarrow cum - P(j)$

Apesar de muito muito interessante a estratégia gulosa não produz sempre soluções ótimas, e existem situações bem simples que o algoritmo é levado a escolher um ítem de maior relação $U(i)/P(i)$ que não faz parte da solução ótima, como no seguinte exemplo extraído de [1], com $n = 3$ e $M = 50$:

n	1	2	3
U	60	100	120
P	10	20	30
U/P	6	5	4
X	0	0	0

Sempre que o algoritmo guloso for usado o ítem número 1 será colocado na Mochila, devido ao seu valor maior que os demais, sendo que este ítem não leva a uma solução. Sendo que o valor ótimo neste exemplo é 220, e os ítens 2 e 3:

n	1	2	3	Max
U	60	100	120	160
P	10	20	30	30
U/P	6	5	4	
X	1	1	0	

n	1	2	3	Max
U	60	100	120	180
P	10	20	30	40
U/P	6	5	4	
X	1	0	1	

n	1	2	3	Max
U	60	100	120	220
P	10	20	30	50
U/P	6	5	4	
X	0	1	1	

5 Comparação entre os Métodos

Neste trabalho fizemos os teste considerando a correlação entre os Pesos e as Utilidades que foram gerados aleatoriamente, conforme citado em [4] e [5]. Os testes foram divididos em:

- Dados Não-Correlacionados - nas instâncias não existe correlação entre os Pesos e as Utilidade de um item. Tais instâncias ilustram as situações onde é razoável admitir que a Utilidade não depende dos Peso, por exemplo quando estamos fazendo uma mudança no caminhão: coisas pequenas podem ser mais valiosas que alguns ítens volumosos. Instâncias não correlacionadas são mais fáceis de resolver, pois existe uma grande variação ente os pesos, tornando mais fácil obter uma Mochila cheia. Fica mais fácil eliminar numerosas variáveis pelo teste do limite ou por relações de dominância.

- Instâncias com Correlação Fraca - aqui a Utilidade é altamente correlacionada com o Peso. Tipicamente a Utilidade difere do Peso por uma faixa pequena. Tais instâncias são mais realistas em gerenciamento, desde que o retorno de um investimento geralmente é proporcional ao investimento somado com alguma variação. Uma alta correlação significa que é mais difícil eliminar variáveis pelo teste de limite. Apesar deste fato instâncias correlacionadas fracamente são mais fáceis de resolver pois existe uma grande faixa de variação de Pesos, tornando mais fácil preencher a Mochila e chegando mais perto da solução ótima.

- Instâncias com Correlação Forte - Tais instâncias correspondem a situações da vida real onde o retorno é uma função linear do investimento mais (ou menos) alguma parte imprecisa em cada projeto. Este tipo de correlação é difícil de resolver por duas razões: 1) Todos os ítens que estão próximo do item de parada tem pesos similares, significando que é muito difícil combina-los de maneira a encher a Mochila. 2) Existe uma grande perda relativa quando removemos ítens de pequeno peso para alocar um item de peso maior. Assim alta correlação são utilizadas para para avaliar a capacidade do algoritmo para resolver problemas difíceis.

Para os experimentos foram usados quatro conjuntos de dados fornecidos pelo colega "David Menoti Gomes". Os conjuntos foram utilizados porque foram gerados seguindo a orientação da referência [4]. Sendo os dois primeiros conjuntos de dados baseado na distribuição de dados não-correlacionados, sendo:

1. (I) Pesos e Utilidades distribuídos na faixa de [1-1000];
2. (II) Pesos e Utilidades distribuídos na faixa de [1-100];

O terceiro conjunto de dados é o conjunto altamente correlacionado, sendo os pesos distribuídos na faixa de [1-100] e a utilidade é o peso acrescido de uma valor constante, $U = P + 10$ (III).
O quarto conjunto de dados é fracamente correlacionado, sendo os pesos distribuídos na faixa de [1-100] e a utilidade calculada como como $U = 1,1 * P$ (IV).
Os experimentos seguiram a seguinte estratégia:

- Para cada n foram feitas 5 execuções, cada uma com 10 amostras, onde a distribuição dos pesos e utilidades é como definido anteriormente

$$n \in 10, 20, 30, 40, 50, 100, 200, 300, 400, 500$$

.

- Cada execução possui uma capacidade da Mochila na faixa de

$$M \in 10\%, 20\%, 30\%, 40\%, 50\%$$

do valor do total dos pesos.

- Foram gerados arquivos com as respostas, formatados da seguinte maneira: (Capacidade da Mochila,Media dos tempos, Maior tempo da amostra, Desvio padrão, No. Elementos).

- Para cada método foram gerados 10 arquivos de saída para cada conjunto de dados, sendo cada saída para um valor de n com cinco execuções variando a capacidade da Mochila, sendo cada execução feita 10 vezes.

- Os tempos estão em segundos.

5.1 Resposta para os Conjuntos de Dados

Os testes foram divididos de acordo com os conjunto dos dados e para os conjuntos foram gerados gráficos comparativos para cada valor dos

$$n \in (10, 20, 30, 40, 50, 100, 200, 300, 400, 500)$$

elementos e para capacidade da mochila variando de

$$M \in (10, 20, 30, 40, 50)$$

do valor total dos pesos. Todos os gráficos estão apresentados em anexo, apresentaremos alguns para apresentar os fatos encontrados nas execuções. Foram gerados tabelas para cada n, sendo que cada tabela apresenta o valor médio do tempo de resposta, o maior tempo gasto e o desvio padrão da amostra, as tabelas também estão em anexo.

Sendo que o tempo de execução do algoritmo Guloso tem se mostrado bastante competitivo quando aplicado aos conjuntos de dados I e II, como podemos observar na figura 10, onde não existe correlação entre os Pesos e Utilidades. O algoritmo de Backtracking também possui um tempo médio bastante competitivo em relação ao guloso. Neste caso ele sempre apresenta o valor ótimo juntamente com o algoritmo dinâmico, podemos verificar esta comparação na figura 7 . Sendo que quando os dados começam a ficar correlacionados, como ocorre nos conjuntos III e IV, o método guloso ainda mantém uma grande competitividade no tempo de execução, porém ele não apresenta sempre o valor ótimo, todas as tabelas estão em Anexo. Podemos verificar na tabela 8 a seguir com os valores acumulados das utilidades para uma execução dos três algoritmo no conjunto de dados III, para $n = (50, 100, 200, 400)$, podemos notar que para este conjunto de dados, a partir de 200 elementos o algoritmo de backtracking não consegue responder num espaço de tempo inferior a 10s, e a execução foi abortada:

Isto também pode ser notado na tabela 9 da execução para o conjunto de dados IV para $n = (50, 100, 200, 400)$, sendo que neste caso o Backtracking não consegue responder a partir de 100 elementos, tendo que ser abortado:

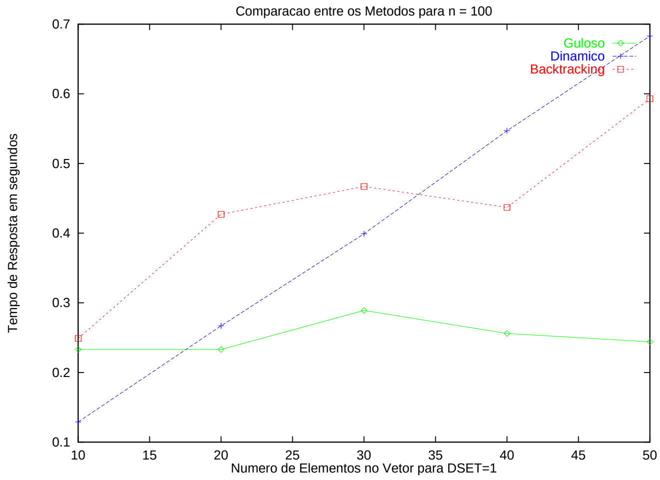

Figura 6: Conjunto de dados I , n=100

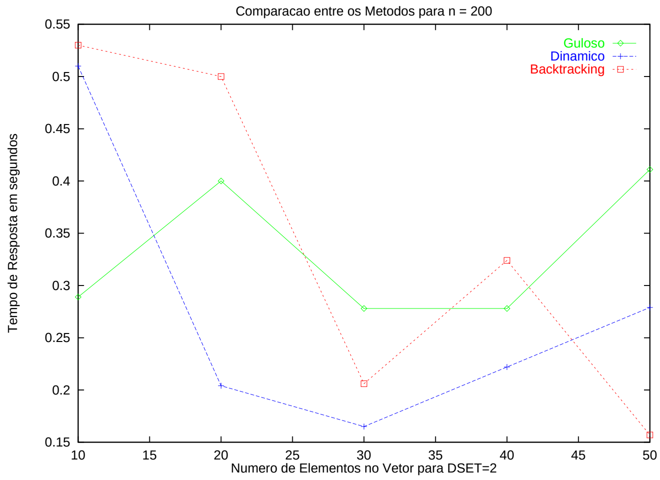

Figura 7: Conjunto de dados II , n=200

No conjunto de dados do tipo I e II o Backtracking consegue responder até 10.000 elementos, já nos conjuntos correlacionados o algoritmo de backtracking não consegue dar uma resposta quando o valor de n chega a 200 para o conjunto III e quando n chega a 100 para o conjunto IV. Nestes casos o algoritmo Dinâmico consegue apresentar melhores resultados. Conforme podemos observar nos gráficos gerados para os diversos valores de n, percebemos que os algoritmo são totalmente dependentes das entradas de dados, sendo que o comportamento de cada algoritmo pode variar bastante para o mesmo conjunto de dados. Sendo o que se apresenta melhor é o Algoritmo Dinâmico, apesar dele levar mais tempo para todas as instâncias é o mais estável com relação a respostas em todos os conjunto de dados.

O que podemos concluir das execuções é que apesar de obtermos implementações interessantes para o problema da Mochila, qualquer dos algoritmos construídos não resolverá de maneira satisfatória o problema. Isso já era previsto, uma vez que o mesmo é NP-Completo.

```
    DIN      BACK     GUL

10  3548.00  3548.00  3407.00  50
20  6368.00  6368.00  6155.00  50
30  9018.00  9018.00  8719.00  50
40 11617.00 11617.00 11382.00  50
50 14151.00 14151.00 13813.00  50

10  7206.00  7206.00  7095.00 100
20 12839.00 12839.00 12652.00 100
30 18235.00 18235.00 18025.00 100
40 23491.00 23491.00 23229.00 100
50 28658.00 28658.00 28353.00 100

10 14515.00          14322.00 200
20 25872.00          25699.00 200
30 36675.00          36453.00 200
40 47200.00          46964.00 200
50 57529.00          57240.00 200

10 29204.00          29061.00 400
20 52097.00          51898.00 400
30 73881.00          73570.00 400
40 95073.00          94710.00 400
50 115910.00        115741.00 400
```

Figura 8: Tabela com tempo de execução para Conjunto de dados III

6 Estruturas de Dados

Para a implementação dos três métodos citados acima, foi construída uma única estrutura de dados que foi utilizadas nas implementações. Construímos um único vetor que armazena os pesos, as utilidades, uma chave que é utilizada na rotina de ordenação, com o objetivo de manter o vetor na ordem decrescente da relação Utilidade/Peso, mantemos também alguns campos que armazenavam a solução que será atualizada para cada método, esta estrutura contém os seguintes ítens:

- Um tipo registro (struct) para cada ítem, contendo o valor da Importância, o Peso, e uma Chave contendo a relação I/P, e três valores para armazenar o resultado para cada método, sendo 1-faz parte da solução, 0-não faz parte da solução: ResultG, ResultB, ResultD.
- Uma estrutura do chamada Vetor, que é um *array* de Ítens, para armazenar os n elementos com suas Utilidades e Pesos.

Na execução do programa podemos definir se a entrada será feita via arquivo ou será gerada internamente, este controle é feito via a variável *LeArquivo*. Se $LeArquivo =$ 1 a entrada é via arquivo, se $LeArquivo = 0$ a entrada será gerada internamente, através do procedimento $GeraVetor$. Neste caso o valor da capacidade da Mochila

```
    DIN      BACK     GUL

10  2442.00  2442.00  2419.00  50
20  4875.00  4875.00  4850.00  50
30  7298.00  7298.00  7280.00  50
40  9723.00  9723.00  9705.00  50
50 12145.00 12145.00 12122.00  50

10  5042.00           5031.00 100
20 10051.00          10047.00 100
30 15054.00          15051.00 100
40 20052.00          20047.00 100
50 25045.00          25035.00 100

10 10213.00          10212.00 200
20 20347.00          20340.00 200
30 30471.00          30469.00 200
40 40575.00          40571.00 200
50 50671.00          50670.00 200

10 20284.00          20283.00 400
20 40427.00          40427.00 400
30 60540.00          60538.00 400
40 80626.00          80625.00 400
50 100686.00        100686.00 400
```

Figura 9: Tabela com tempo de execução para Conjunto de dados IV

é calculado como metade do total dos pesos, conforme arquivo fonte em anexo.
O valor dos pesos e das utilidades, quando gerados internamente, são gerados numa faixa controlada pela constante N que pode ser alterada. A capacidade do vetor é definido pela constante $MaxTam$, sendo que este valor é o limite e a quantidade dos objeto na Mochila são definidos pela variável $NoElementos$.

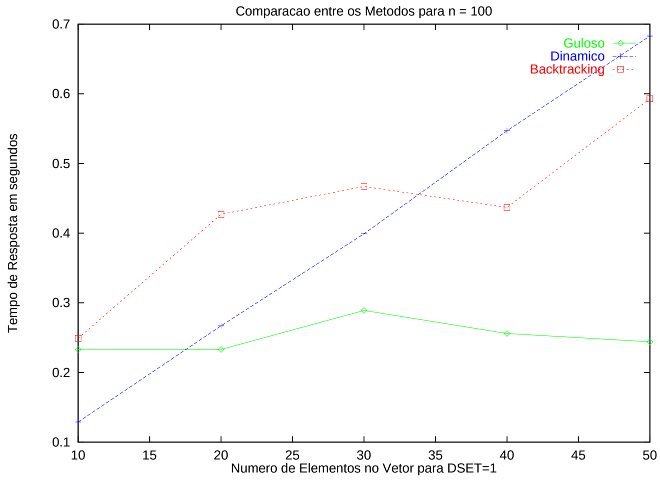

Figura 10: Conjunto de dados I , n=100

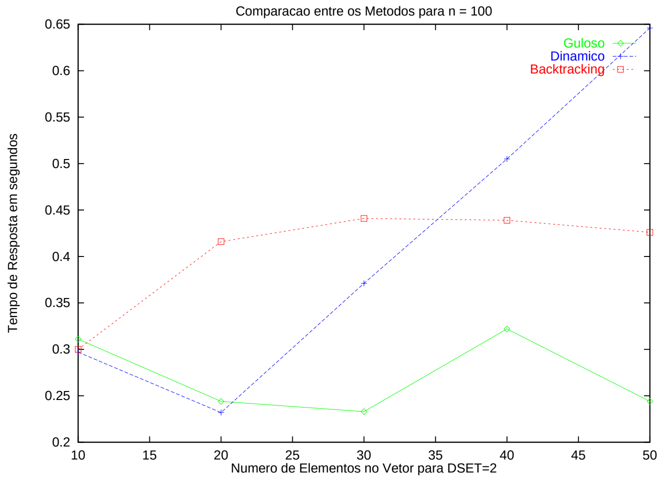

Figura 11: Conjunto de dados II , n=100

Referências

[1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[2] Pierluigi Crescenzi and Viggo Kann. A compendium of NP-optimization problems
http://www.nada.kth.se/ viggo/wwwcompendium/wwwcompendium.html.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[4] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[5] David Pisinger. *Algorithms fo Knapsack Problems*. PhD thesis, Department of Computer Science, University of Copenhagen, February 1995.

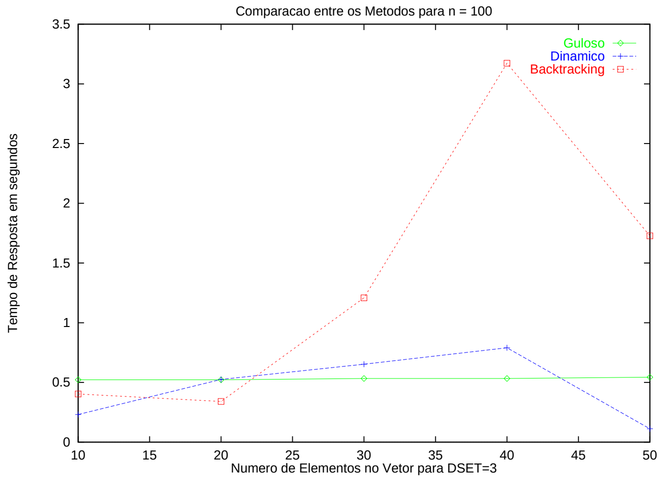

Figura 12: Conjunto de dados III , n=100

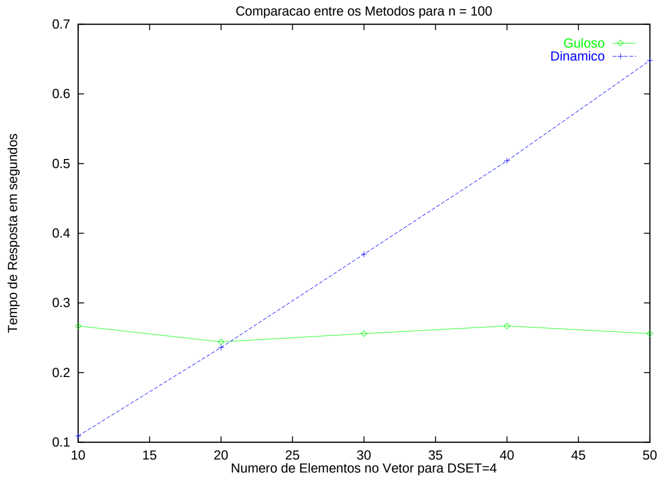

Figura 13: Conjunto de dados IV , n=100

[6] Nivio Ziviane. *Projeto de Algoritmos: com implementações em Pascal e C*. Pioneira Thomson Learning, 2ed., 2004.

```
 #define MaxTam  1000  //Tamanho máximo do Vetor
 #define N 500          //Maior valor Aleatorio
 #define NoElementos 300//Quantidade de Elementos para a Mochila
 #define LeArquivo 1   // 1 - via arquivo, 0-Gerado Randomicamente
 typedef int Valor;
 typedef struct Item {
     Valor Importancia;
     Valor Peso;
     float Chave;
     short ResultG;
     short ResultB;
     short ResultD;
 } Item;
 typedef int Indice;//Vetor de Registros
 typedef Item Vetor[MaxTam+1];
 Vetor Origem;
```

Figura 14: Trecho de código das principais definições

A Código Fonte

```
/* ========================================================================
 2 Trabalho Pratico de PAA - 2004-1
 aluno - Ruiter Braga Caldas
 Progama que Executa tres metodos para resolver o problema da mochila:
 Método Guloso e Backtracking desenvolvidos a aprtir do algoritmo
 proposto em - E. Horowitz e S. Sahni, Fundamentals of Computer Algorithms,
 Computer Science Press, 1978.
 Método Dinâmico desenvolvido a partir dos códigos propostos em:
 http://www-cse.uta.edu/~holder/courses/cse2320/lectures/l15/node12.html
 http://www.mpi-sb.mpg.de/~rybal/armc-live-termin/node5.html
====================================================================== */
 #include <stdlib.h>
 #include <stdio.h>
 #include <sys/time.h>
 #include <limits.h>
 #include "time.h"
 #define MaxTam  1000       //Tamanho máximo do Vetor
 #define N 500              //Maior valor Aleatorio
 #define NoElementos 300    //Quantidade de Elementos para a Mochila
 #define LeArquivo 1        // 1 - via arquivo, 0 - Gerado Randomicamente
 typedef int Valor;
/* ========================================================================
   Estrutura tipo Registro que armazena os Pesos, Importancias,
   Chave, e Resultados. Os resultados são armazenados de acordo com o metodo
  ====================================================================== */
 typedef struct Item {
   Valor Importancia;
```

```
    Valor Peso;
    float Chave;
    short ResultG;
    short ResultB;
    short ResultD;
  } Item;

  typedef int Indice;

  //Vetor de Registros
  typedef Item Vetor[MaxTam+1];
   Vetor Origem;

  //Peso Final do Método Backtracking
  float Tpeso,Timportancia;
  Indice i, n, k;

  //Capacidade da Mochila
  int M;
  //Arquivos de Entrada e Saida
  FILE* fp;
  FILE* fs;
/* ========================================================================
    Função QuickSort disponibilizada no livro
    do Prof. Nivio Ziviane.(Projeto de Algoritmos/2004).
    Foi Alterada para fazer a ordenacao Crescente.
   ======================================================================= */
  void Particao(Indice Esq, Indice Dir, Indice *i, Indice *j, Item *A)
  { Item x, w;
    *i = Esq; *j = Dir;
    x = A[(*i + *j) / 2];   /* obtem o pivo x */
    do
      { while (x.Chave < A[*i].Chave) (*i)++;
        while (x.Chave > A[*j].Chave) (*j)--;
        if (*i <= *j)
        { w = A[*i]; A[*i] = A[*j]; A[*j] = w;
          (*i)++; (*j)--;
        }
      } while (*i <= *j);
  }

  void Ordena(Indice Esq, Indice Dir, Item *A)
  { Indice i, j;
    Particao(Esq, Dir, &i, &j, A);
    if (Esq < j) Ordena(Esq, j, A);
    if (i < Dir) Ordena(i, Dir, A);
  }
  void QuickSort(Item *A, Indice *n)
  { Ordena(1, *n, A);
  }
```

```
/* ========================================================================
    O código foi copiado da biblioteca random
    de Eric Roberts.(The Art and Science of C)
   ======================================================================= */
  int RandomInteger (int low, int high)
  {
      double k;
      double d;
      d = (double) rand () / ((double) RAND_MAX + 1);
      k = d * (high - low + 1);
      return (int)(low + k);
  }
/* ========================================================================
    Função que faz a impressão dos Pesos e Importancias no Métod Guloso
    A impressão é dirigida pelo Valor de ResultG
   ======================================================================= */
  void ImprimeGuloso(Item *V, Indice *n)
  { int Tp;
    int Ti;
      Tp=0;
      Ti=0;
      for (i = 1; i <= *n; i++)
      {
      if (V[i].ResultG)
          fprintf(fs," %d %d\n",V[i].Peso,V[i].Importancia);
  }
  }
/* ========================================================================
    Função que faz a impressão dos Pesos e Imports. no Métod Backtracking
    A impressão é dirigida pelo Valor de ResulB
   ======================================================================= */
  void ImprimeBack(Item *V, Indice *n, double Tp, double Ti)
  {
  for (i = 1; i <= *n; i++)
      {
      if (V[i].ResultB)
          fprintf(fs,"%d %d\n",V[i].Peso,V[i].Importancia);
      }
  }
/* ========================================================================
    Função que faz a impressão dos Pesos e Importancias no Métod Dinâmico
    A impressão é dirigida pelo Valor do ResulD
   ======================================================================= */
  void ImprimeDinamico(Item *V, Indice *n)
  {int Tp;
   int Ti;
      Tp=0;
      Ti=0;
      for (i = 1; i <= *n; i++)
      {
```

```
            if (V[i].ResultD)
                fprintf(fs," %d %d\n",V[i].Peso,V[i].Importancia);
        }
    }

    void CopiaDois(int Fonte[],Item *Destino, Indice *n)
    { for (i = 1; i <= *n; i++)
         Destino[i].ResultB = Fonte[i];
        }
/* ======================================================================
      Função que Gera Randomicamente o Vetor com os Pesos e Importancias,
      faz o calculo da Chave=Importancia/Peso e Iniciaiza os Resultados.
      O valor de M eh a metade do total dos pesos.
     ====================================================================== */
    void GeraVetor(Vetor A, Indice n,int *M)
    { int  i;
      Valor pesoTotal;
      pesoTotal=0;
      for (i = 1; i <= n; i++) {
        A[i].Importancia = RandomInteger(1,N);
        A[i].Peso = RandomInteger(1,N);
        A[i].ResultG = 0;
        A[i].ResultB = 0;
        A[i].ResultD = 0;
        pesoTotal = pesoTotal + A[i].Peso;
        A[i].Chave =(double)A[i].Importancia / (double)A[i].Peso;};
        //Calcula o Valor da Mochila com 1/2 Pesos
        *M=pesoTotal*1/2;
    }
/* ======================================================================
      Executa o Metodo Guloso
     ====================================================================== */
    void Guloso(Vetor A, Indice n, int M)
    { int  i,j, CM;
      CM=M;
      for (i = 1; i <= n; i++)
      {
        if (A[i].Peso > CM){
            break;
            }
        A[i].ResultG = 1;
        CM = CM - A[i].Peso;
       }
      j=i+1;
      while (j <= n)
      {
         if (A[j].Peso <= CM)
            {
              CM = CM - A[j].Peso;
              A[j].ResultG = 1;
```

```
        }
      j=j+1;
    }
  }
/* =======================================================================
    Executa o Metodo Dinâmico
   ======================================================================= */
 //funcao Limite para o método
  float Bound(Vetor A,float p, float w, int k, float M)
  {
  float b,c;
  b = p;
  c = w;
  for (i = k + 1; i <= n; i++)
      {
      c = c + A[i].Peso;
      if (c <= M){ b = b + A[i].Importancia;
          }
      else return (b + (1 -((c - M)/A[i].Peso))*A[i].Importancia);
      }
  return(b);
  }
  void BKnap(float M, Indice n, Vetor A, float *fw, float *fp)
  { int  k, Y[n];
    float cw,cp;

    cw = 0.0;
    cp = 0.0;
    k = 1;
    *fp = -1;

    while (1)
    {
      while (( k <= n ) && ( (cw + A[k].Peso) <= M ))
      {
          cw = cw + A[k].Peso;
          cp = cp + A[k].Importancia;
          Y[k]= 1;
          k = k + 1;
       }
      if (k > n)
      { *fp = cp;
        *fw = cw;
        k = n;
        CopiaDois(Y,A,&n);
      }
      else Y[k]=0;
      while (Bound(A,cp,cw,k,M) <= *fp)
      {
           while (k != 0 && Y[k] != 1)
```

```
                    {
                    k = k - 1;
                    };
           if ( k == 0 ) return;
           Y[k] = 0;
           cw = cw - A[k].Peso;
           cp = cp - A[k].Importancia;
      }//while (Bound)
      k = k + 1;
     }//while(1)
  }
/* ====================================================================
    Executa o Metodo Dinâmico
   ==================================================================== */
  void Dinamico(Item *A, Indice n, int M)
  { int j;
    int c;
    int **a;
    Indice i;

    a = (int **) calloc(n+1,sizeof(int *));
    for( i=0; i<n+1; i++ )
         a[i] = (int *) calloc( M+1, sizeof(int));
    //IniciaMatriz
    for (i = 0; i <= n; i++)
      for (j = 0; j <= M; j++) a[i][j] = -1;
    //Inicializa Linha
    for(c = 0; c <= M; c++)a[0][c] = 0;
    //Inicializa Coluna
    for(i = 0; i <= n; i++)a[i][0] = 0;
    for(i = 1; i <= n; i++)
    {
       for (c = 1; c <= M; c++)
       {
        if (A[i].Peso <= c)
        {
          if ((A[i].Importancia + a[i-1][c-A[i].Peso]) > a[i-1][c] )
          {
            a[i][c] = A[i].Importancia + a[i-1][c-A[i].Peso];
          }
          else
            {
            a[i][c] = a[i-1][c];
            }
         }
        else
        {
          a[i][c] = a[i-1][c];
        }
      }
```

```
    }
  //Calcula os elementos que fazem parte da Resposta
    i=n;
    j=M;
    while ((i > 0) && (j > 0))
    {
      if (a[i][j] != a[i-1][j])
          {
          A[i].ResultD = 1;
          j=j-A[i].Peso;
          i--;
          }
      else{
            A[i].ResultD = 0;
            i--;
          }
    }
    for( i=0; i < n+1; i++ )
       free(a[i]);
    free(a);
  }

/* =========================================================================
    Função Principal. Deve receber como chamada a primeira letra do Metodo
    G - Guloso, B - Backtracking, D - Dinamico.
    A variável LeArquivo deve ser setada antes:
        1 - Le os dados do arquivo chamado avaliacao.tp2
        0 - Gera os dados num vetor de Pesos e Importancias
    Quando os dados sao gerados, a quantidade de elementos no vetor de
    pesos/importancias fica determinada pela variavel NoElementos=200.
    O peso da Mochila( M ) fica determinado pela metade das soma dos pesos
    gerados.
    O tamanho do maior vetor eh determinado pela variável MaxTam=1000
    O faixa de valore para os pesos/importancias eh determinado pela
    variavel N=500.
   ======================================================================= */
  int main(int argc, char * argv[])
  {
    srand((int)time(NULL));
    //Inicializa a Capacidade da Mochila
    M=0;
    //Verifica se a entrada sera via Arquivo ou Geracao Randomica
    if (LeArquivo)
    {
      if ((fp = fopen("avaliacao.tp2", "r")) == NULL)
             fprintf(stderr, "ERRO na Abertura do Arquivo de Entrada\n");
  else
      {
          printf("Arquivo de Entrada aberto com Sucesso.\n");
          i = 1 ;
```

```
          //Le o numero de elementos do Vetor
          fscanf(fp, "%d", &n);
          fscanf(fp, "%d", &M);
          while(!feof(fp) && i <=n)
          {
              /*  */
              fscanf(fp, "%d", &Origem[i].Peso);
              fscanf(fp, "%d", &Origem[i].Importancia);
                  Origem[i].Chave =(double)Origem[i].Importancia /
                   (double)Origem[i].Peso;
              i++;
          }
      }
    }
    else
      {
        //Numero de elementos do Vetor
        n=NoElementos;
        // Gera o Vetor com os Pesos, Importancia
        // e Calcula a Capacidade da Mochila
        GeraVetor(Origem,n,&M);
       }
    if (argc == 1) fprintf(stderr,"ERRO: Precisa Informar um Metodo (G/B/D) \n");
    else{
    //Ordena em Orden Crescente de Imp/Peso
    QuickSort(Origem,&n);
    switch (*argv[1]){
    case 'G'://Chama Metodo Guloso
             fprintf(stderr, "Guloso\n");
             startTimer();
             Guloso(Origem,n,M);
             finishTimer();
             showTimes();
             //Abre o Arquivo de Saida
             if ((fs = fopen("saidag", "w")) == NULL)
             fprintf(stderr, "ERRO na Abertura do Arquivo de Saida\n");
             ImprimeGuloso(Origem,&n);
             break;
    case 'B'://Chama Metodo Backtracking
            fprintf(stderr, "Backtracking\n");
            Tpeso = 0.0;
            Timportancia = 0.0;
            startTimer();
            BKnap(M,n,Origem,&Tpeso,&Timportancia);
            finishTimer();
            showTimes();
            //Abre o Arquivo de Saida
            if ((fs = fopen("saidab", "w")) == NULL)
            fprintf(stderr, "ERRO na Abertura do Arquivo de Saida\n");
            ImprimeBack(Origem,&n,Tpeso,Timportancia);
```

```
            break;
    case 'D'://Chama Metodo Dinâmico
             fprintf(stderr, "Dinamico\n");
             startTimer();
             Dinamico(Origem, n, M);
             finishTimer();
             showTimes();
             //Abre o Arquivo de Saida
             if ((fs = fopen("saidad", "w")) == NULL)
             fprintf(stderr, "ERRO na Abertura do Arquivo de Saida\n");
             ImprimeDinamico(Origem, &n);
             break;
    default :fprintf(stderr,"ERRO: Método Não informado (G/B/D) \n");
    }}
    fclose(fp);
    fclose(fs);
    return 0;
  }

```

B Tabelas com tempos de execução

B.1 Método Guloso

B.1.1 Conjunto I

```
 M media  max  desv.  n
10 0.256 0.700 0.059  10
20 0.278 0.500 0.024  10
30 0.244 0.500 0.059  10
40 0.256 0.600 0.059  10
50 0.422 1.000 0.236  10
10 0.667 1.000 0.354  20
20 0.333 1.000 0.141  20
30 0.511 1.000 0.519  20
40 0.589 1.000 0.436  20
50 0.622 1.000 0.342  20
10 0.344 1.000 0.153  30
20 0.489 1.000 0.306  30
30 0.511 1.000 0.118  30
40 0.500 1.000 0.530  30
50 0.489 1.000 0.306  30
10 0.767 1.000 0.247  40
20 0.411 1.000 0.224  40
30 0.233 0.500 0.035  40
40 0.422 1.000 0.236  40
50 0.367 1.000 0.177  40
10 0.589 1.000 0.436  50
20 0.400 1.000 0.212  50
```

```
30 0.300 0.500 0.106  50
40 0.411 1.000 0.625  50
50 0.311 1.000 0.118  50
10 0.322 0.500 0.130 100
20 0.233 0.500 0.035 100
30 0.244 0.400 0.047 100
40 0.322 0.500 0.024 100
50 0.244 0.600 0.047 100
10 0.289 0.500 0.094 200
20 0.400 0.600 0.106 200
30 0.389 0.900 0.012 200
40 0.400 0.800 0.106 200
50 0.444 0.700 0.059 200
10 0.367 0.900 0.071 300
20 0.344 0.600 0.047 300
30 0.478 0.700 0.130 300
40 0.456 0.900 0.059 300
50 0.411 0.700 0.012 300
10 0.422 0.800 0.401 400
20 0.511 0.900 0.118 400
30 0.433 0.700 0.035 400
40 0.433 0.700 0.035 400
50 0.501 0.700 0.107 400
10 0.433 0.700 0.035 500
20 0.489 0.700 0.012 500
30 0.644 0.900 0.259 500
40 0.500 0.800 0.000 500
50 0.533 0.700 0.035 500
```

B.1.2 Conjunto II

```
 M media  max  desv.  n
10 0.244 0.500 0.047  10
20 0.256 0.600 0.059  10
30 0.256 0.500 0.059  10
40 0.233 0.500 0.035  10
50 0.322 1.000 0.719  10
10 0.244 0.600 0.047  20
20 0.767 1.000 0.601  20
30 0.667 1.000 0.354  20
40 0.356 1.000 0.165  20
50 0.500 1.000 0.318  20
10 0.489 1.000 0.306  30
20 0.600 1.000 0.424  30
30 0.511 1.000 0.519  30
40 0.667 1.000 0.495  30
50 0.422 1.000 0.024  30
10 0.356 1.000 0.684  40
20 0.400 1.000 0.212  40
30 0.578 1.000 0.401  40
```

```
40 0.289 0.700 0.094  40
50 0.500 1.000 0.318  40
10 0.411 1.000 0.224  50
20 0.467 1.000 0.283  50
30 0.489 1.000 0.306  50
40 0.578 1.000 0.401  50
50 0.356 1.000 0.165  50
10 0.222 0.400 0.024 100
20 0.233 0.400 0.035 100
30 0.322 0.500 0.130 100
40 0.267 0.600 0.071 100
50 0.278 0.600 0.082 100
10 0.378 0.600 0.024 200
20 0.289 0.500 0.012 200
30 0.300 0.500 0.000 200
40 0.378 0.600 0.024 200
50 0.367 0.800 0.071 200
10 0.333 0.600 0.035 300
20 0.411 0.600 0.094 300
30 0.433 0.800 0.035 300
40 0.356 0.600 0.059 300
50 0.467 0.700 0.141 300
10 0.467 0.800 0.071 400
20 0.422 0.700 0.024 400
30 0.388 0.600 0.210 400
40 0.556 1.000 0.059 400
50 0.456 0.700 0.059 400
10 0.422 0.600 0.024 500
20 0.524 0.700 0.132 500
30 0.478 0.700 0.082 500
40 0.511 0.800 0.012 500
50 0.722 1.000 0.236 500
```

B.1.3 Conjunto III

```
 M media  max desv.  n
10 0.222 0.400 0.024  10
20 0.233 0.500 0.035  10
30 0.244 0.500 0.047  10
40 0.244 0.500 0.047  10
50 0.244 0.500 0.047  10
10 0.578 1.000 0.448  20
20 0.489 1.000 0.306  20
30 0.522 1.000 0.342  20
40 0.422 1.000 0.236  20
50 0.489 1.000 0.306  20
10 0.522 1.000 0.236  30
20 0.256 0.500 0.059  30
30 0.278 0.600 0.024  30
40 0.244 0.500 0.047  30
```

```
50 0.511 1.000 0.519  30
10 0.567 1.000 0.460  40
20 0.522 1.000 0.130  40
30 0.422 1.000 0.613  40
40 0.400 1.000 0.212  40
50 0.400 1.000 0.212  40
10 0.278 0.600 0.082  50
20 0.411 1.000 0.625  50
30 0.311 1.000 0.118  50
40 0.344 1.000 0.695  50
50 0.489 1.000 0.306  50
10 0.222 0.400 0.024 100
20 0.300 0.500 0.106 100
30 0.233 0.400 0.035 100
40 0.222 0.400 0.024 100
50 0.322 0.600 0.130 100
10 0.267 0.500 0.071 200
20 0.289 0.500 0.094 200
30 0.400 0.500 0.000 200
40 0.322 0.500 0.024 200
50 0.322 0.500 0.024 200
10 0.433 0.600 0.071 300
20 0.411 0.800 0.012 300
30 0.367 0.600 0.035 300
40 0.467 0.700 0.141 300
50 0.456 0.900 0.059 300
10 0.400 0.600 0.000 400
20 0.467 0.700 0.141 400
30 0.500 0.900 0.106 400
40 0.478 0.700 0.082 400
50 0.500 0.700 0.212 400
10 0.567 0.900 0.177 500
20 0.489 0.800 0.012 500
30 0.522 0.800 0.024 500
40 0.593 0.800 0.099 500
50 0.556 0.800 0.059 500
```

B.1.4 Conjunto IV

```
 M media  max  desv.  n
10 0.267 0.500 0.071  10
20 0.233 0.400 0.035  10
30 0.256 0.500 0.047  10
40 0.322 1.000 0.130  10
50 0.411 1.000 0.625  10
10 0.400 1.000 0.636  20
20 0.578 1.000 0.448  20
30 0.422 1.000 0.613  20
40 0.667 1.000 0.495  20
50 0.400 1.000 0.212  20
```

```
10 0.267 0.500 0.071  30
20 0.489 1.000 0.306  30
30 0.222 0.400 0.024  30
40 0.367 1.000 0.071  30
50 0.322 1.000 0.130  30
10 0.678 1.000 0.342  40
20 0.344 1.000 0.047  40
30 0.600 1.000 0.424  40
40 0.311 1.000 0.118  40
50 0.522 1.000 0.236  40
10 0.344 1.000 0.153  50
20 0.322 1.000 0.130  50
30 0.433 1.000 0.247  50
40 0.267 0.600 0.071  50
50 0.311 1.000 0.731  50
10 0.267 0.500 0.247 100
20 0.278 0.600 0.082 100
30 0.244 0.500 0.047 100
40 0.244 0.400 0.059 100
50 0.356 0.800 0.165 100
10 0.256 0.400 0.047 200
20 0.311 0.600 0.012 200
30 0.422 0.700 0.236 200
40 0.311 0.700 0.118 200
50 0.333 0.600 0.035 200
10 0.444 0.700 0.153 300
20 0.344 0.600 0.047 300
30 0.356 0.600 0.047 300
40 0.511 0.700 0.224 300
50 0.389 0.600 0.012 300
10 0.333 0.500 0.035 400
20 0.522 0.700 0.024 400
30 0.422 0.700 0.024 400
40 0.400 0.600 0.000 400
50 0.633 0.900 0.283 400
10 0.648 0.900 0.570 500
20 0.768 0.900 0.034 500
30 0.689 0.800 0.012 500
40 0.457 0.700 0.046 500
50 0.500 0.700 0.000 500
```

B.2 Método Dinâmico

B.2.1 Conjunto I

```
 M media  max  desv.  n
10 0.407 1.000 0.324  10
20 0.234 0.349 0.044  10
30 0.361 0.508 0.012  10
40 0.482 0.623 0.113  10
```

```
50 0.514 0.683 0.044  10
10 0.373 0.466 0.028  20
20 0.760 0.967 0.032  20
30 0.233 0.944 0.111  20
40 0.194 0.250 0.010  20
50 0.252 0.319 0.009  20
10 0.672 0.976 0.154  30
20 0.211 0.253 0.023  30
30 0.330 0.389 0.027  30
40 0.448 0.524 0.031  30
50 0.616 0.721 0.001  30
10 0.210 0.239 0.006  40
20 0.445 0.509 0.045  40
30 0.693 0.784 0.089  40
40 0.856 0.977 0.083  40
50 0.158 0.203 0.026  40
10 0.323 0.351 0.029  50
20 0.556 0.685 0.478  50
30 0.227 0.976 0.134  50
40 0.225 0.982 0.090  50
50 0.160 0.179 0.014  50
10 0.125 0.136 0.011 100
20 0.256 0.279 0.024 100
30 0.388 0.430 0.045 100
40 0.555 0.620 0.068 100
50 0.654 0.711 0.061 100
10 0.518 0.570 0.002 200
20 0.112 0.122 0.011 200
30 0.157 0.171 0.000 200
40 0.214 0.230 0.006 200
50 0.271 0.304 0.007 200
10 0.119 0.130 0.008 300
20 0.235 0.274 0.012 300
30 0.351 0.370 0.018 300
40 0.474 0.500 0.025 300
50 0.590 0.629 0.029 300
10 0.211 0.217 0.015 400
20 0.417 0.432 0.031 400
30 0.637 0.678 0.052 400
40 0.864 0.911 0.076 400
50 1.492 1.959 0.528 400
10 0.322 0.336 0.005 500
20 0.635 0.675 0.010 500
30 0.961 0.989 0.030 500
40 1.303 1.383 0.026 500
50 1.636 1.692 0.059 500
```

B.2.2 Conjunto II

```
 M media  max  desv.  n
10 0.226 0.360 0.079  10
20 0.374 0.460 0.027  10
30 0.481 0.600 0.084  10
40 0.508 0.670 0.040  10
50 0.552 0.660 0.051  10
10 0.450 0.630 0.042  20
20 0.712 1.000 0.104  20
30 0.132 0.173 0.011  20
40 0.175 0.225 0.015  20
50 0.227 0.275 0.026  20
10 0.717 0.930 0.225  30
20 0.191 0.245 0.005  30
30 0.302 0.397 0.019  30
40 0.378 0.488 0.001  30
50 0.475 0.612 0.003  30
10 0.183 0.230 0.016  40
20 0.355 0.463 0.016  40
30 0.537 0.687 0.021  40
40 0.756 0.931 0.034  40
50 0.625 0.994 0.314  40
10 0.258 0.305 0.013  50
20 0.539 0.609 0.045  50
30 0.815 0.930 0.052  50
40 0.115 0.124 0.009  50
50 0.149 0.170 0.012  50
10 0.395 0.968 0.299 100
20 0.237 0.258 0.015 100
30 0.368 0.399 0.025 100
40 0.508 0.555 0.050 100
50 0.651 0.692 0.039 100
10 0.514 0.544 0.004 200
20 0.202 0.953 0.099 200
30 0.165 0.173 0.001 200
40 0.221 0.238 0.003 200
50 0.283 0.293 0.000 200
10 0.118 0.124 0.005 300
20 0.243 0.254 0.001 300
30 0.367 0.380 0.010 300
40 0.490 0.508 0.015 300
50 0.611 0.634 0.015 300
10 0.213 0.226 0.000 400
20 0.435 0.464 0.007 400
30 0.645 0.678 0.001 400
40 0.865 0.912 0.008 400
50 0.108 0.115 0.000 400
10 0.337 0.353 0.007 500
20 0.677 0.713 0.016 500
30 0.299 0.990 0.207 500
```

```
40 0.135 0.141 0.004 500
50 0.171 0.179 0.005 500
```

B.2.3 Conjunto III

```
 M media  max  desv.  n
10 0.239 0.430 0.054  10
20 0.402 0.600 0.008  10
30 0.520 0.750 0.021  10
40 0.578 0.910 0.019  10
50 0.533 0.700 0.071  10
10 0.467 0.640 0.004  20
20 0.622 0.930 0.547  20
30 0.136 0.157 0.008  20
40 0.180 0.213 0.015  20
50 0.230 0.267 0.033  20
10 0.485 1.000 0.472  30
20 0.210 0.268 0.017  30
30 0.317 0.402 0.001  30
40 0.426 0.545 0.043  30
50 0.530 0.663 0.045  30
10 0.175 0.254 0.010  40
20 0.352 0.509 0.067  40
30 0.524 0.742 0.088  40
40 0.625 0.783 0.033  40
50 0.720 0.998 0.049  40
10 0.259 0.302 0.015  50
20 0.532 0.665 0.141  50
30 0.817 0.881 0.037  50
40 0.209 0.953 0.095  50
50 0.147 0.167 0.022  50
10 0.209 0.983 0.099 100
20 0.239 0.274 0.014 100
30 0.382 0.419 0.032 100
40 0.524 0.582 0.061 100
50 0.668 0.718 0.042 100
10 0.501 0.532 0.001 200
20 0.106 0.113 0.002 200
30 0.163 0.172 0.001 200
40 0.219 0.232 0.000 200
50 0.273 0.290 0.000 200
10 0.119 0.124 0.000 300
20 0.243 0.250 0.000 300
30 0.367 0.382 0.001 300
40 0.489 0.502 0.006 300
50 0.612 0.629 0.003 300
10 0.214 0.225 0.001 400
20 0.427 0.451 0.006 400
30 0.644 0.675 0.008 400
40 0.862 0.908 0.008 400
```

```
50 0.109 0.113 0.000 400
10 0.337 0.347 0.004 500
20 0.674 0.691 0.009 500
30 0.301 0.999 0.209 500
40 0.136 0.140 0.002 500
50 0.171 0.175 0.002 500
```

B.2.4 Conjunto IV

```
 M media  max  desv.  n
10 0.231 0.400 0.062  10
20 0.389 0.480 0.065  10
30 0.520 0.590 0.074  10
40 0.573 0.760 0.198  10
50 0.712 0.850 0.146  10
10 0.472 0.640 0.066  20
20 0.795 0.960 0.016  20
30 0.144 0.176 0.025  20
40 0.183 0.232 0.020  20
50 0.227 0.283 0.027  20
10 0.389 0.980 0.585  30
20 0.206 0.277 0.023  30
30 0.312 0.412 0.033  30
40 0.436 0.559 0.060  30
50 0.521 0.677 0.059  30
10 0.166 0.210 0.021  40
20 0.345 0.420 0.050  40
30 0.505 0.630 0.061  40
40 0.676 0.821 0.090  40
50 0.700 0.972 0.084  40
10 0.260 0.334 0.025  50
20 0.519 0.638 0.046  50
30 0.836 0.960 0.096  50
40 0.210 0.971 0.115  50
50 0.143 0.160 0.012  50
10 0.112 0.125 0.009 100
20 0.241 0.261 0.000 100
30 0.372 0.410 0.007 100
40 0.517 0.570 0.010 100
50 0.665 0.721 0.002 100
10 0.509 0.534 0.007 200
20 0.108 0.112 0.001 200
30 0.165 0.171 0.001 200
40 0.218 0.229 0.007 200
50 0.275 0.299 0.011 200
10 0.119 0.129 0.002 300
20 0.239 0.256 0.012 300
30 0.356 0.389 0.006 300
40 0.476 0.517 0.012 300
50 0.597 0.644 0.015 300
```

```
10 0.216 0.244 0.001 400
20 0.430 0.455 0.011 400
30 0.643 0.660 0.015 400
40 0.856 0.889 0.027 400
50 0.107 0.111 0.003 400
10 0.332 0.337 0.002 500
20 0.667 0.679 0.010 500
30 0.495 0.996 0.530 500
40 0.133 0.135 0.001 500
50 0.168 0.171 0.003 500
```

B.3 Método Backtracking

B.3.1 Conjunto I

```
 M media  max  desv.  n
10 0.444 0.800 0.047  10
20 0.500 0.800 0.106  10
30 0.522 0.900 0.024  10
40 0.522 1.000 0.130  10
50 0.522 1.000 0.130  10
10 0.568 0.900 0.178  20
20 0.578 1.000 0.189  20
30 0.644 1.000 0.271  20
40 0.672 0.900 0.183  20
50 0.637 0.900 0.067  20
10 0.606 0.800 0.112  30
20 0.297 0.800 0.534  30
30 0.506 0.900 0.206  30
40 0.634 1.000 0.037  30
50 0.493 1.000 0.113  30
10 0.414 1.000 0.280  40
20 0.383 0.900 0.548  40
30 0.283 0.900 0.654  40
40 0.158 0.200 0.019  40
50 0.328 0.900 0.220  40
10 0.541 1.000 0.381  50
20 0.197 0.310 0.049  50
30 0.221 0.460 0.044  50
40 0.258 0.660 0.082  50
50 0.218 0.370 0.082  50
10 0.247 0.450 0.025 100
20 0.431 0.700 0.181 100
30 0.487 0.670 0.156 100
40 0.454 0.690 0.005 100
50 0.595 0.890 0.143 100
10 0.513 0.830 0.177 200
20 0.506 1.000 0.237 200
30 0.172 0.650 0.051 200
40 0.148 0.224 0.046 200
```

```
50 0.196 0.276 0.016 200
10 0.281 0.930 0.161 300
20 0.187 0.390 0.068 300
30 0.220 0.296 0.057 300
40 0.269 0.368 0.033 300
50 0.300 0.390 0.050 300
10 0.207 0.345 0.004 400
20 0.285 0.481 0.078 400
30 0.355 0.432 0.064 400
40 0.515 0.650 0.143 400
50 0.541 0.693 0.132 400
10 0.239 0.326 0.057 500
20 0.490 0.599 0.106 500
30 0.616 0.920 0.322 500
40 0.497 0.884 0.410 500
50 0.733 0.967 0.194 500
```

B.3.2 Conjunto II

```
 M media  max  desv.  n
10 0.422 0.800 0.130  10
20 0.456 0.700 0.059  10
30 0.533 0.800 0.141  10
40 0.467 0.700 0.177  10
50 0.333 0.600 0.035  10
10 0.480 0.800 0.085  20
20 0.508 1.000 0.220  20
30 0.611 1.000 0.224  20
40 0.622 0.900 0.130  20
50 0.643 1.000 0.378  20
10 0.722 1.000 0.295  30
20 0.617 0.900 0.018  30
30 0.310 0.900 0.127  30
40 0.582 1.000 0.448  30
50 0.683 1.000 0.194  30
10 0.602 1.000 0.210  40
20 0.563 0.800 0.145  40
30 0.377 1.000 0.343  40
40 0.317 0.800 0.513  40
50 0.392 0.900 0.289  40
10 0.337 0.900 0.209  50
20 0.286 1.000 0.123  50
30 0.271 0.800 0.211  50
40 0.164 0.220 0.048  50
50 0.252 0.380 0.019  50
10 0.299 0.490 0.115 100
20 0.381 0.600 0.126 100
30 0.481 0.600 0.266 100
40 0.437 0.860 0.007 100
50 0.394 0.540 0.078 100
```

```
10 0.568 0.860 0.097 200
20 0.501 0.780 0.126 200
30 0.297 0.950 0.194 200
40 0.335 0.990 0.219 200
50 0.153 0.235 0.035 200
10 0.525 0.970 0.397 300
20 0.291 0.960 0.144 300
30 0.203 0.246 0.004 300
40 0.239 0.348 0.020 300
50 0.332 0.446 0.081 300
10 0.220 0.282 0.002 400
20 0.287 0.469 0.055 400
30 0.309 0.395 0.046 400
40 0.445 0.602 0.016 400
50 0.511 0.753 0.041 400
10 0.273 0.391 0.117 500
20 0.367 0.603 0.108 500
30 0.450 0.552 0.046 500
40 0.645 0.909 0.280 500
50 0.688 0.763 0.025 500
```

B.3.3 Conjunto III

```
 M media  max  desv.  n
10 0.600 0.900 0.000  10
20 0.516 1.000 0.123  10
30 0.569 0.800 0.033  10
40 0.413 0.700 0.226  10
50 0.427 1.000 0.396  10
10 0.373 1.000 0.152  20
20 0.356 0.800 0.154  20
30 0.378 0.990 0.242  20
40 0.394 0.700 0.324  20
50 0.423 1.000 0.305  20
10 0.355 0.520 0.101  30
20 0.343 0.650 0.326  30
30 0.310 0.700 0.026  30
40 0.322 0.840 0.550  30
50 0.338 0.770 0.295  30
10 0.291 0.470 0.169  40
20 0.434 0.961 0.559  40
30 0.452 0.970 0.352  40
40 0.364 0.787 0.006  40
50 0.356 0.930 0.235  40
10 0.505 0.800 0.313  50
20 0.495 0.840 0.387  50
30 0.256 0.489 0.155  50
40 0.385 0.916 0.022  50
50 0.398 0.723 0.200  50
10 0.406 0.736 0.343 100
```

```
20 0.342 0.812 0.498 100
30 1.218 7.314 1.094 100
40 3.197 11.623 2.181 100
50 1.906 11.841 1.004 100
10 41.021 108.697 42.888 200
```

B.3.4 Conjunto IV

```
  M media  max  desv.  n
10 0.700 1.000 0.106  10
20 0.234 0.800 0.121  10
30 0.189 0.300 0.086  10
40 0.318 1.000 0.002  10
50 0.274 0.400 0.027  10
10 0.447 1.000 0.028  20
20 0.479 1.000 0.012  20
30 0.469 0.890 0.306  20
40 0.435 0.910 0.503  20
50 0.373 0.730 0.026  20
10 0.447 0.970 0.050  30
20 0.417 0.850 0.143  30
30 0.337 0.773 0.462  30
40 0.512 0.900 0.030  30
50 0.400 0.900 0.530  30
10 0.270 0.716 0.159  40
20 0.474 0.883 0.111  40
30 0.375 0.754 0.035  40
40 0.496 0.800 0.144  40
50 0.420 0.754 0.274  40
10 0.474 0.910 0.130  50
20 0.299 0.649 0.085  50
30 0.395 0.800 0.222  50
40 0.331 0.528 0.142  50
50 0.358 0.740 0.405  50
10 0.588 1.553 0.390 100
```

C Tabelas com Utilidade Acumulada

C.1 Conjunto I

```
    Dinamico Back.    Guloso
10 14276.00 14276.00 13463.00  10
20 20503.00 20503.00 20314.00  10
30 26288.00 26288.00 25955.00  10
40 31148.00 31148.00 31122.00  10
50 34319.00 34319.00 33893.00  10

10 33096.00 33096.00 32207.00  20
20 48454.00 48454.00 47708.00  20
```

```
30 59570.00 59570.00 59331.00  20
40 69157.00 69157.00 68721.00  20
50 76959.00 76959.00 76761.00  20

10  46778.00  46778.00   46244.00  30
20  66220.00  66220.00   65214.00  30
30  81565.00  81565.00   80927.00  30
40  94349.00  94349.00   94082.00  30
50 105394.00 105394.00  105237.00  30

10  61772.00  61772.00  60735.00  40
20  88281.00  88281.00  87548.00  40
30 108874.00 108874.00 108189.00  40
40 126150.00 126150.00 125697.00  40
50 141115.00 141115.00 140701.00  40

10  74270.00  74270.00  73779.00  50
20 108386.00 108386.00 107406.00  50
30 134295.00 134295.00 133837.00  50
40 155956.00 155956.00 155182.00  50
50 173403.00 173403.00 173076.00  50

10 159587.00 159587.00 159117.00 100
20 225225.00 225225.00 224695.00 100
30 276070.00 276070.00 275470.00 100
40 319273.00 319273.00 318824.00 100
50 353836.00 353836.00 353349.00 100

10 329200.00 329200.00 328781.00 200
20 467250.00 467250.00 466546.00 200
30 573983.00 573983.00 573761.00 200
40 661586.00 661586.00 661056.00 200
50 735671.00 735671.00 735181.00 200

10  491996.00  491996.00  491460.00 300
20  696382.00  696382.00  695955.00 300
30  851705.00  851705.00  851307.00 300
40  983043.00  983043.00  982816.00 300
50 1094717.00 1094717.00 1094409.00 300

10  636679.00  636679.00  636239.00 400
20  907682.00  907682.00  907330.00 400
30 1117518.00 1117518.00 1117112.00 400
40 1290594.00 1290594.00 1290068.00 400
50 1437606.00 1437606.00 1437342.00 400

10  837932.00  837932.00  837633.00 500
20 1176953.00 1176953.00 1176508.00 500
30 1435879.00 1435879.00 1435541.00 500
40 1651514.00 1651514.00 1651357.00 500
```

```
50 1832626.00 1832626.00 1832218.00 500
```

C.2 Conjunto I

```
   Dinamico Back.    Guloso
10 1295.00 1295.00 1277.00  10
20 2083.00 2083.00 2044.00  10
30 2609.00 2609.00 2536.00  10
40 3083.00 3083.00 3016.00  10
50 3462.00 3462.00 3448.00  10

10 2885.00 2885.00 2854.00  20
20 4355.00 4355.00 4302.00  20
30 5480.00 5480.00 5431.00  20
40 6349.00 6349.00 6291.00  20
50 7072.00 7072.00 6985.00  20

10  5153.00  5153.00  5059.00  30
20  7268.00  7268.00  7197.00  30
30  8826.00  8826.00  8751.00  30
40 10141.00 10141.00 10101.00  30
50 11243.00 11243.00 11213.00  30

10  5443.00  5443.00  5377.00  40
20  8194.00  8194.00  8186.00  40
30 10455.00 10455.00 10411.00  40
40 12383.00 12383.00 12358.00  40
50 14074.00 14074.00 14047.00  40

10  7883.00  7883.00  7831.00  50
20 11243.00 11243.00 11194.00  50
30 13785.00 13785.00 13734.00  50
40 15914.00 15914.00 15880.00  50
50 17681.00 17681.00 17646.00  50

10 16965.00 16965.00 16893.00 100
20 24188.00 24188.00 24129.00 100
30 29518.00 29518.00 29450.00 100
40 34010.00 34010.00 33975.00 100
50 37893.00 37893.00 37847.00 100

10 32672.00 32672.00 32631.00 200
20 46526.00 46526.00 46482.00 200
30 57293.00 57293.00 57248.00 200
40 66397.00 66397.00 66372.00 200
50 73981.00 73981.00 73936.00 200

10  46954.00  46954.00  46922.00 300
20  67436.00  67436.00  67374.00 300
30  83481.00  83481.00  83448.00 300
```

```
40  96750.00  96750.00  96708.00 300
50 108015.00 108015.00 107981.00 300

10  63963.00  63963.00  63933.00 400
20  91576.00  91576.00  91525.00 400
30 112954.00 112954.00 112920.00 400
40 130763.00 130763.00 130727.00 400
50 145819.00 145819.00 145795.00 400

10  81070.00  81070.00  81031.00 500
20 115021.00 115021.00 114992.00 500
30 141652.00 141652.00 141619.00 500
40 164171.00 164171.00 164132.00 500
50 183215.00 183215.00 183186.00 500
```

C.3 Conjunto I

```
   Dinamico Back.    Guloso
10 5042.00 5042.00 5031.00 100
10  638.00  638.00  536.00  10
20 1223.00 1223.00 1125.00  10
30 1787.00 1787.00 1623.00  10
40 2313.00 2313.00 2110.00  10
50 2840.00 2840.00 2674.00  10

10 1380.00 1380.00 1243.00  20
20 2530.00 2530.00 2338.00  20
30 3620.00 3620.00 3419.00  20
40 4656.00 4656.00 4476.00  20
50 5691.00 5691.00 5349.00  20

10 2166.00 2166.00 2041.00  30
20 3935.00 3935.00 3671.00  30
30 5674.00 5674.00 5506.00  30
40 7339.00 7339.00 7019.00  30
50 9011.00 9011.00 8771.00  30

10  2859.00  2859.00  2680.00  40
20  5165.00  5165.00  4918.00  40
30  7397.00  7397.00  7243.00  40
40  9513.00  9513.00  9170.00  40
50 11613.00 11613.00 11192.00  40

10  3548.00  3548.00  3407.00  50
20  6368.00  6368.00  6155.00  50
30  9018.00  9018.00  8719.00  50
40 11617.00 11617.00 11382.00  50
50 14151.00 14151.00 13813.00  50

10  7206.00  7206.00  7095.00 100
```

```
20 12839.00 12839.00 12652.00 100
30 18235.00 18235.00 18025.00 100
40 23491.00 23491.00 23229.00 100
50 28658.00 28658.00 28353.00 100

10 14515.00          14322.00 200
20 25872.00          25699.00 200
30 36675.00          36453.00 200
40 47200.00          46964.00 200
50 57529.00          57240.00 200

10 21859.00          21708.00 300
20 39047.00          38885.00 300
30 55396.00          55128.00 300
40 71334.00          71023.00 300
50 87036.00          86856.00 300

10  29204.00          29061.00 400
20  52097.00          51898.00 400
30  73881.00          73570.00 400
40  95073.00          94710.00 400
50 115910.00         115741.00 400

10  36710.00          36632.00 500
20  65513.00          65315.00 500
30  92992.00          92724.00 500
40 119814.00         119524.00 500
50 146195.00         145895.00 500
```

C.4 Conjunto I

```
   Dinamico Back.    Guloso
10  445.00  445.00  396.00  10
20  932.00  932.00  765.00  10
30 1404.00 1404.00 1300.00  10
40 1873.00 1873.00 1755.00  10
50 2342.00 2342.00 2259.00  10

10 1004.00 1004.00  943.00  20
20 2002.00 2002.00 1921.00  20
30 2999.00 2999.00 2917.00  20
40 3998.00 3998.00 3924.00  20
50 4998.00 4998.00 4968.00  20

10 1572.00 1572.00 1550.00  30
20 3141.00 3141.00 3107.00  30
30 4709.00 4709.00 4678.00  30
40 6273.00 6273.00 6236.00  30
50 7835.00 7835.00 7794.00  30

```

```
10 1953.00 1953.00 1913.00  40
20 3900.00 3900.00 3838.00  40
30 5841.00 5841.00 5826.00  40
40 7783.00 7783.00 7755.00  40
50 9721.00 9721.00 9683.00  40

10  2442.00  2442.00  2419.00  50
20  4875.00  4875.00  4850.00  50
30  7298.00  7298.00  7280.00  50
40  9723.00  9723.00  9705.00  50
50 12145.00 12145.00 12122.00  50

10  5042.00          5031.00 100
20 10051.00          10047.00 100
30 15054.00          15051.00 100
40 20052.00          20047.00 100
50 25045.00          25035.00 100
10 10213.00          10212.00 200
20 20347.00          20340.00 200
30 30471.00          30469.00 200
40 40575.00          40571.00 200
50 50671.00          50670.00 200

10 15045.00          15043.00 300
20 29979.00          29978.00 300
30 44890.00          44888.00 300
40 59783.00          59782.00 300
50 74663.00          74659.00 300

10  20284.00          20283.00 400
20  40427.00          40427.00 400
30  60540.00          60538.00 400
40  80626.00          80625.00 400
50 100686.00         100686.00 400

10  25368.00          25368.00 500
20  50556.00          50556.00 500
30  75714.00          75714.00 500
40 100834.00         100834.00 500
50 125928.00         125928.00 500
```