Armazenamento de arquivos grandes em DVDs
MAC5758 - Introdução ao Escalonamento e Aplicações
Professor: Alfredo Goldman

Viviane Teles de Lucca Maranhão

13 de dezembro de 2009

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Neste trabalho estudaremos o problema *Bin-packing* unidimensional que é um problema NP-difícil:

"Seja L = $< a_1, a_2, ..., a_n >$ uma lista com números no intervalo $(0, 1]$, e uma seqüência de *bins* (recipentes) com capacidade unitária $B_1, B_2, ...$ Encontre uma atribuição dos números aos *bins* de modo que em nenhum *bin* a soma dos números atribuídos a ele seja maior que 1 e tal que o número de *bins* utilizados, isto é, com pelo menos um número atribuído a ele, é minimizado."

Vamos apresentar algumas heurísticas *online* para sua resolução como *Best-Fit, Next-Fit, Worst-Fit...* e heurísticas *offline* como *Best-Fit Decreasing*, e apresentaremos o desempenho de algumas dessas heurísticas no caso médio e no pior caso, fazendo uma breve revisão bibliográfica numa pequena parte da grande quantidade de material que já existe publicada sobre o assunto.

Após isso partiremos para o seguinte problema:

"Dados n arquivos de tamanhos t_1, t_2, ... , t_n, distribuí-los em x DVDs de tamanho C de modo que x seja o menor possível."

E este será solucionado utilizando as mesmas técnicas apresentadas para o *Bin-packing*. Implementaremos os métodos estudados em um programa feito em *Python* no qual o usuário entra com uma lista de arquivos, escolhe o método que deseja utilizar e o programa calcula quantos DVDs serão necessários para armazená-los e sugere uma maneira de gravá-los nesses DVDs.

Terminamos o trabalho analisando o desempenho do programa elaborado para alguns casos de teste (com solução ótima conhecida) e comparamos os resultados com o esperado pelo que foi estudado na revisão bibliográfica.

1 Introdução

A contínua melhora na qualidade e velocidade das conexões à internet torna cada vez mais ampla quantidade de dados trocada através de programas de compartilhamento ou outros meios, inclusive em alguns momentos gerando um problema de armazenamento para tal informação.

Ao contrário dos discos rígidos que são alterados com bastante frequência, em CDs ou DVDs é costume alterar pouco as informações neles gravadas, ressaltando que alguns tipos só permitem gravar uma única vez de modo que é conveniente gravar utilizando de maneira ótima a mídia disponivel, minimizando o número de CDs ou DVDs necessários.

Neste trabalho nos preocuparemos com a utilização de DVDs, sendo que o problema pode ser facilmente adaptado para gravação de arquivos em CDs, ou até mesmo em uma escala menor como em disquetes. Podemos então formular o problema como:

"Dados n arquivos de tamanhos t_1, t_2, ... , t_n, distribuí-los em x DVDs de tamanho C de modo que x seja o menor possivel"

Para evitar complicações desnecessárias, vamos supor $t_i \leq t_x, i = 1, 2...n$

Outros problemas semelhantes ao anterior são citados por [1] como:

- Um carpinteiro que pretende construir uma estrutura complexa mas só dispõe de toras de madeira de tamanho fixo e quer comprar o menor número de toras para construir uma certa estrutura e cortar todos os tamanhos de madeira necessários para ela;
- Uma indústria de aço que produz barras de aço de diferentes tamanhos para reforçar o concreto;
- Uma emissora de televisão que deseja colocar nos intervalos comerciais o máximo de propagandas possível.

Todos os problemas acima são problemas de empacotamento e, dadas pequenas modificações necessárias, podem ser reduzidos ao problema *bin packing*, ou problema do empacotamento unidimensional, que é definido da seguinte forma:

"Seja L = $< a_1, a_2, ..., a_n >$ uma lista com números no intervalo $(0, 1]$, e uma seqüência de *bins* (recipentes) com capacidade unitária $B_1, B_2, ...$ Encontre uma atribuição dos números aos *bins* de modo que em nenhum *bin* a soma dos números atribuídos a ele seja maior que 1, e tal que o número de *bins* utilizados, isto é, com pelo menos um número atribuído a ele, é minimizado."

E este é um problema NP-difícil, o que significa que é improvável que exista um algoritmo exato que possa resolver toda e qualquer instância em tempo polinomial ou pseudopolinomial.

2 Objetivo

O objetivo deste trabalho é fazer um programa em Python que proponha uma maneira eficiente de gravar arquivos passados pelo usuário no menor número possível de DVDs com uma capacidade dada.

Para tanto, a princípio estudaremos o problema do "bin packing" e algumas das possíveis técnicas para sua resolução e em seguida aplicaremos o conteúdo estudado na elaboração do programa.

3 Revisão Bibliográfica

3.1 Heurísticas

Consideramos duas classes de algoritmos para problemas de empacotamento em geral, a classe *online* e a classe *offline*. Os algoritmos chamados offline são aqueles em que todos os dados da instância são conhecidos pelo algoritmo de antemão enquanto na classe de algoritmos chamados online os dados não são conhecidos *online*. Existe uma grande variedade de heurísticas para a resolução do *bin-packing* e não vamos cobrir todas elas neste trabalho, mais informações podem ser encontradas em [3] e [1].

Consideraremos ainda que os recipientes B_1,B_2,...B_k são criados sequencialmente, que $s(a_i)$ é o tamanho do item a_i, e finalmente que *nivel(B)* é a soma dos tamanhos de todos os itens em B. Com isto posto, podemos partir para as heurísticas de resolução do problema. Optamos por preservar os nomes das heurísticas em inglês por ser a maneira que elas são frequentemente encontradas.

3.1.1 Next-Fit (NF)

O NF é possívelmente a mais simples das heurísticas do *bin-packing* unidimensional, na qual somente o último recipiente utilizado (B_j, onde j é o maior índice tal que B_j não é vazio) fica ativo, estado que denominaremos por "aberto". Um item a_i pode ser alocado em B_j se satisfizer a seguinte condição:

$$s(a_i) \leq 1 - nivel(B_j)$$

Caso a_i possa ser alocado em B_j, B_j permanece aberto. Caso contrário, B_j é fechado e a_i é adicionado em um novo recipiente, B_{j+1} que agora é o novo recipiente aberto.

Notemos que nesta heurística apenas um recipente fica aberto durante o processo de alocação dos items e que esta é uma heurística *online*.

3.1.2 First-Fit (FF)

Nesta heurística verificamos a condição

$$s(a_i) \leq 1 - nivel(B_j)$$

para todos os recipientes B_j, parando assim que a condição for satisfeita (e alocando a_i neste recipiente), ou abrindo um novo recipiente no qual a_i será alocado caso a condição não se verifique para nenhum dos B_j abertos até o momento.

O FF é uma heurística *online* similar à anterior, com a diferença que varre os recipientes parcialmente cheios, permitindo que existam dois ou mais recipientes abertos simultaneamente. Como a varredura é feita sempre em ordem crescente nos recipientes abertos, o FF tende a concentrar os itens nos primeiros recipientes, ([4])

3.1.3 Best-Fit (BF)

No BF a cada alocação todos os recipientes são avaliados e o que apresentar a menor sobra de espaço após a alocação é selecionado, sendo que o desempate é feito a favor do menor índice. Notamos que isso é equivalente a escolher o recipiente com o maior *nivel*. Assim como nos casos anteriores, um novo recipiente é aberto quando não é possível alocar a_i em nenhum dos recipientes anteriores e esta também é uma heurística online.

3.1.4 Worst-Fit (WF) e Almost-Worst-Fit (AWF)

O WF aplica o teste inverso ao BF, de modo que o recipiente selecionado é o que resta o maior espaço após a alocação, ou seja, o recipiente de menor *nivel*, e o desempate também é feito a favor do recipiente de menor índice. O AWF, uma pequena modificação do primeiro, aloca a_i no recipiente que após a alocação deste apresente a segunda maior sobra de espaço (desempatando como no primeiro), a menos que haja só um recipiente no qual seja possível alocar a_i, e naturalmente, abrindo um recipiente novo caso não seja possível alocar a_i em nenhum dos recipientes abertos.

3.1.5 Any-Fit (AF) e Almost-Any-Fit (AAF)

Estas duas classificações concluem nossa exposição de heurísticas *online*. Um algoritmo para o *bin-packing* é um algoritmo *Any-Fit* caso um recipiente novo seja aberto somente na condição de não ser possível alocar o item atual em nenhum dos recipientes abertos parcialmente cheios no momento. Um algoritmo *Almost-Any-Fit* só coloca um item no recipiente com o menor *nivel* caso existam dois recipientes com o mesmo *nivel* ou caso este seja o único recipiente no qual é possível alocar tal item. Essas classificações são úteis na hora de analizar o desempenho de tais algoritmos.

3.1.6 First-Fit-Decreasing (FFD) e Best-Fit-Decreasing (BFD)

Estas duas heurísticas consistem em ordenar o vetor com os itens em ordem não crescente de tamanho antes de aplicar o FF ou o BF, respectivamente. Por causa do conhecimento prévio sobre toda a instância, esses algoritmos são *offline*, e justamente pela possibilidade de ordenação prévia garantem melhor desempenho que seus respectivos algoritmos *online*.

3.1.7 Exemplo

Vamos considerar a seguinte lista de objetos $L = (a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$ com seus tamanhos $S(L) = (\frac{1}{2}, \frac{3}{4}, \frac{3}{8}, \frac{2}{5}, \frac{2}{3}, \frac{1}{8}, \frac{3}{5}, \frac{1}{4})$ mostrada na figura 1.

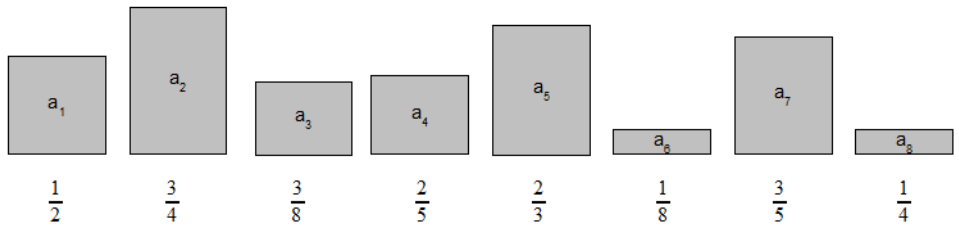

Figura 1: Exemplo de uma lista de objetos

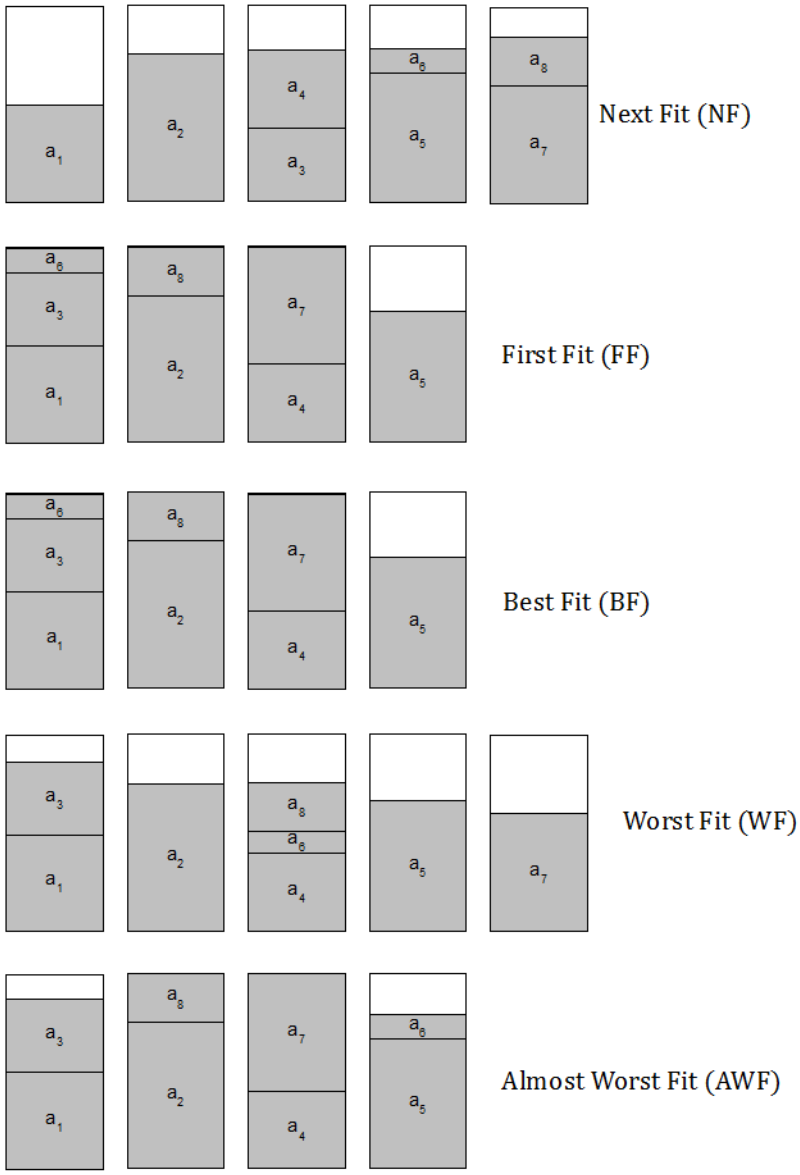

Figura 2: Alocação usando heurísticas online

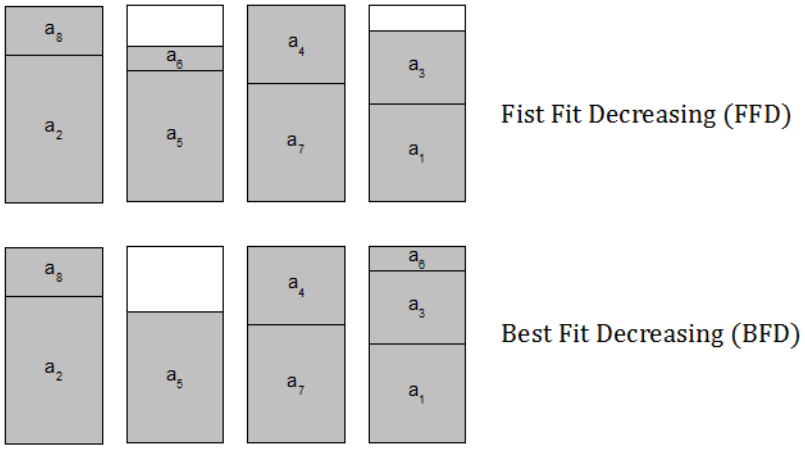

Figura 3: Alocação usando heurísticas offline

Aplicando cada uma das heurísticas apresentadas chegamos ao resultado final que pode ser visto nas figuras 2 e 3. Vemos que o número de recipientes utilizados por cada uma das heurísticas e a maneira na qual os itens são armazenados não é necessariamente igual, desta forma, um breve estudo sobre o desempenho de cada uma delas pode nos ajudar a escolher uma heurística adequada para determinada situação.

3.2 Desempenho de pior caso

Vamos utilizar a seguinte notação: Dado um algoritmo A e uma lista de objetos L para um problema de minimização, denotamos por $A(L)$ o valor da solução devolvida pelo algoritmo A aplicado à instância L e denotamos por $OPT(L)$ o correspondente valor para uma solução ótima de L. Definimos $R_A(L)$ por $\frac{A(L)}{OPT(L)}$ e definimos a razão absoluta de performace de pior caso R_A do algoritimo A por:

$$R_A \equiv inf\{r \geq 1 : R_A(L) \leq r \; \forall \; L\}$$

Podemos também considerar aproximações assintóticas que são comumente utilizadas. Neste caso definimos para um algoritmo A sua razão

assintótica de pior caso R_A^∞ por:
$R_A^\infty \equiv inf\{r \geq 1 : \text{para algum } N > 0, R_A(L) \leq r \; \forall \; L \text{ com } OPT(L) \geq N\}$

Adicionalmente, se para as instâncias nas quais todos dos itens tem tamanho no máximo α definimos analogamente as razões de tamanho limitado de performace: $R_A(\alpha)$ e $R_A^\infty(\alpha)$. Os resultados apresentandos a seguir são encontrados em maiores detalhes em [1] e [3].

Next-Fit

O NF pode ser implementado em tempo linear, tem $R_{NF}^\infty = 2$ e $R_{NF}^\infty(\alpha) = \frac{1}{1-\alpha}$, $\alpha < \frac{1}{2}$

First-Fit

O FF tem um consumo de tempo $O(nlogn)$, com n o tamanho da instância, tem $R_{FF}^\infty \cong 1.69103$ e para $\frac{1}{m+1} < \alpha \leq \frac{1}{m}$,$R_{FF}^\infty(\alpha) = 1.7$ se $m = 1$ e $R_{FF}^\infty(\alpha) = 1 + \frac{1}{m}$, se $m \geq 2$

Best-Fit

O BF, assim com o FF tem um consumo de tempo $O(nlogn)$ e tem $R_{BF}^\infty(\alpha) = R_{FF}^\infty(\alpha)$

Worst-Fit

Para o WF temos $R_{WF}^\infty(\alpha) = R_{NF}^\infty(\alpha)$, $0 < \alpha \leq 1$

Almost-Wost-Fit

Nesta modificação do WF vale $R_{AWF}^\infty(\alpha) = R_{FF}^\infty(\alpha)$, $0 < \alpha \leq 1$

Any-Fit e Almost-Any-Fit

Mais genericamente temos os seguintes resultados para todo α, $0 < \alpha \leq 1$

- Se A é um algoritmo AF, então $R_{FF}^\infty(\alpha) \leq R_A^\infty(\alpha) \leq R_{NF}^\infty(\alpha)$,
- Se A é um algoritmo AAF, então $R_A^\infty(\alpha) = R_{FF}^\infty(\alpha)$,

First-Fit-Decreasing e Best-Fit-Decreasing

Temos $R_{FFD}^\infty = R_{BFD}^\infty = \frac{11}{9} = 1.222...$ e ambos consomem tempo $O(nlogn)$.

Temos ainda que, de uma maneira mais geral, para um algoritmo A que ordene os itens em ordem não crescente e depois aplique uma regra Any-Fit vale:

- $\frac{11}{9} \leq R_A^{\infty} \leq \frac{5}{4}$ e
- $\frac{1}{m+2} - \frac{2}{m(m+1)(m+2)} \leq R_A^{\infty}(\alpha) \leq \frac{1}{m+2}$, com $m = \lceil \frac{1}{\alpha} \rceil$

3.3 Desempenho no caso médio

Algumas vezes podemos estar interessados no desempenho no caso médio, um caso que possivelmente ocorrerá na maior parte das vezes ao invés de ter um limite superior para o tempo de execução que pode acaber sendo muito pessimista em certos casos.

Vamos utilizar a seguinte notação: Dada uma lista com n elementos L_n que seguem uma dada distrbuição F com média μ e variância σ^2, e um algoritmo A de resolução do *bin-packing* definimos

$$\bar{R_A^n}(F) \equiv E[R_A(L_n)] = E\left[\frac{A(L_n)}{OPT(L_n)}\right]$$

Seja ainda $s(L_n)$ a somatória de todos os tamanhos dos itens de L_n e $W_A(L_n) \equiv A(L_n) - s(L_n)$ que é o total de espaço desperdiçado nos recipientes através do empacotamento por A. Definimos o espaço desperdiçado esperado por

$$\bar{W_A^n}(L_n) \equiv E[A(L_n) - s(L_n)]$$

Definimos finalmente

$$\bar{R_A^{\infty}}(F) \equiv lim_{n\to\infty}\bar{R_A^n}(F)$$

que é a razão assintótica esperada para A sob F.

Além disso, vamos trabalhar com a seguinte notação assintótica:

- Dizemos que $f(n) = O(g(n))$ se $\exists c > 0$ e n_0 tal que $f(n) \leq c \cdot g(n)$ para todo $n > n_0$
- $f(n) = \Omega(g(n))$ se $\exists c > 0$ e n_0 tal que $f(n) \geq c \cdot g(n)$ para todo $n > n_0$
- $f(n) = \Theta(g(n))$ se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

A seguir, alguns resultados de [3] e [6], assumindo F a distribuição uniforme $U(0, 1)$.

Next-Fit
$\bar{R}^{\infty}_{NF}(U[0,1]) = \frac{4}{3}$ e $\bar{W}^{n}_{NF} \leq \frac{n}{32}$

First-Fit
$\bar{R}^{\infty}_{FF}(U[0,1]) = 1$ e $\bar{W}^{n}_{FF} \leq \Theta(n^{\frac{2}{3}})$

Best-Fit
$\bar{R}^{\infty}_{FF}(U[0,1]) = 1$ e $\bar{W}^{n}_{FF} \leq \Theta(\sqrt{n}log^{\frac{3}{4}}n)$

Worst-Fit
$\bar{R}^{n}_{WF} \leq \Omega(\sqrt{nlogn})$. Este resultado vale para todo algoritmo aberto (que não sabe o valor de n de antemão) *online*.

First-Fit-Decreasing e Best-Fit-Decreasing
Temos que para $A \in \{FFD, BFD\}$ $\bar{W}^{n}_{A} = \Theta(\sqrt{n})$

4 Implementação e Descrição do Programa

A linguagem utilizada na elaboração do programa foi Python 2.6 com a biblioteca gráfica wxpython versão 2.8.10.1. A escolha da linguagem se deu por sua facilidade.

Separamos o programa em dois arquivos:

- **organizador.py** Interface gráfica para o usuário entrar com as informações: tamanho da mídia, lista de arquivos e método desejado para resolução, e que após rodar o método apresenta uma sugestão de gravação.
- **metodos.py** Contém a implementação das heurísticas de empacotamento citadas anteriormente neste trabalho.

4.1 organizador.py

A interface gráfica está toda dentro da classe Organizador. Comentários sobre a implementação dessa classe podem ser encontrados no código fonte em A.1, por isso nos concentraremos na lógica de funcionamento do programa.

Os campos que o usuário pode alterar são os seguintes:

- **Radio box com os métodos** Na qual o usuário escolhe o método que quer utilizar. Por padrão selecionamos o Next-Fit.
- **Caixa de texto com o tamanho da mídia** Onde o usuário escreve o valor em megabytes. Valores inválidos contendo caracteres não numéricos (com a excessão de um único '.') são ignorados e neste caso o valor utilizado é a capacidade padrão de um DVD (4.3 Gb)
- **Lista com os arquivos** Ao clicar em "Adicionar" é aberto um seletor de arquivos com o qual o usuário pode acrescentar arquivos na lista, inclusive selecionando mais de uma vez. Ao inserir duas vezes o mesmo arquivo consideramos que o usuário deseja gravá-lo duas vezes, por isso ele não é excluído da lista. Para remover um arquivo da lista basta o usuário clicar neste arquivo e em seguida em "Remover".

Cada arquivo inserido na lista é apresentado com seu nome e tamanho para o usuário, e tem esses valores respectivamente acrescentados às variáveis self.itens e self.tamanhos, sendo que este último será passado para o método escolhido pelo usuário. Optamos por trabalhar com os tamanhos dos arquivos em bytes para utilizarmos valores sempre inteiros evitando assim problemas com arredondamentos do float.

Quando o usuário clica em "Iniciar" verificamos o método escolhido pelo usuário e chamamos esta função com os parâmetros necessários. Todos os métodos devolvem uma lista de recipientes, e cada recipiente é uma lista com os tamanhos dos itens nele alocados. Com isto, buscamos na lista de itens (com os nomes dos arquivos) os itens correspondentes aos tamanhos e apresentamos no campo de texto com a sugestão de gravação a resposta obtida, separando por dvd e apresentando nos campos inferiores o tempo de execução do método e o número total de dvds.

As figuras 4 e 5 mostram screenshots do programa.

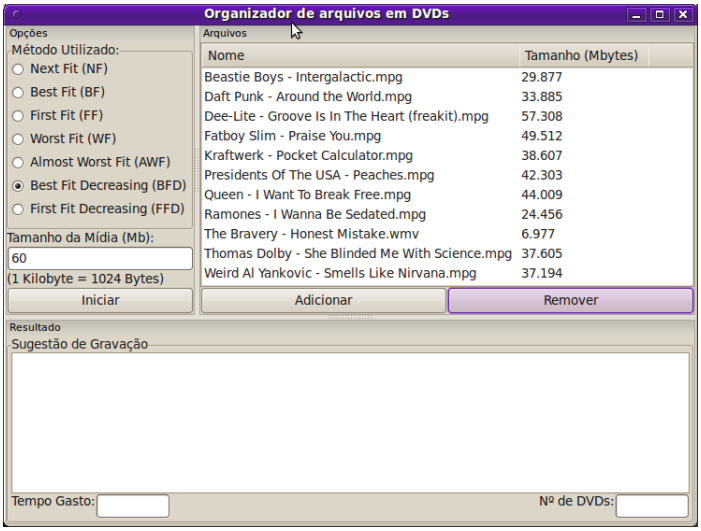

Figura 4: Tela do programa com arquivos selecionados

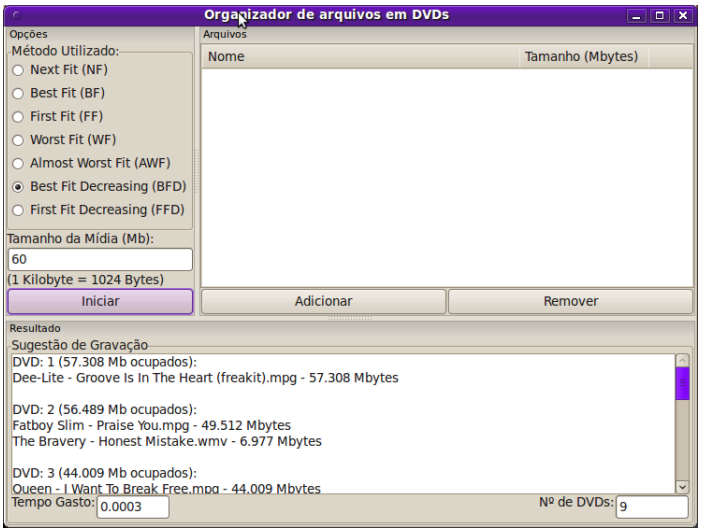

Figura 5: Tela do programa com sugestão de gravação

4.2 metodos.py

Todos os métodos recebem como entrada duas variáveis: uma lista com os tamanhos dos itens e a capacidade da mídia (ambos em bytes).

Para que seja possível utilizar as heurísitcas em uma máquina que possua python, mas não possua a biblioteca wxpython instalada, também foi implementada uma chamada por linha de comando, na qual os parâmetros recebidos são:

- -l ou --lista: Nome do arquivo com a capacidade na primeira linha e a lista com os tamanhos dos itens separados por espaços.
- -n ou --nf: Faz o empacotamento utilizando next-fit.
- -b ou --bf: Faz o empacotamento utilizando best-fit.
- -f ou --ff: Faz o empacotamento utilizando first-fit.
- -w ou --wf: Faz o empacotamento utilizando worst-fit.
- -a ou --awf: Faz o empacotamento utilizando almost-worst-fit.
- -d ou --ffd: Faz o empacotamento utilizando first-fit-decreasing.
- -g ou --bfd: Faz o empacotamento utilizando best-fit-decreasing.
- -i ou --imprime: Imprime os elementos de cada dvd
- -j ou --imprime2: Imprime o nivel de cada dvds

Para imprimir a saída na tela temos a função imprime, que funciona de maneira semelhante à interface gráfica, mas que por não possuir os nomes dos arquivos imprime só os tamanhos na tela, e também permite ao usuário escolher ao quantidade de informação do dvd que deseja imprimir (tamanho utilizado, disposição os itens alocados).

Aproveitando a estrutura do python, cada heurística começa com uma lista contendo uma lista vazia (sem nenhum recipiente, sem nenhum item). Ao aplicar a regra de decisão de cada heurística, considerando o nível de um recipiente dado, decidimos se o item vai ser adicionado a um recipiente já existente, ou se abrirá um recipiente novo. Procedendo desta maneira para todos os tamanhos chegamos ao resultado final, que é uma lista onde cada

posição é um recipiente e contém um lista com os tamanhos nele alocados, e devolvemos esta variável para o programa.

Os níveis dos recipientes abertos são armazenados em uma lista que é inicializada com zero na primeira posição, na qual cada posição corresponde ao nível do recipiente de mesmo índice, de modo que a cada passo, ao invés de fazer uma somatória (com excessão do next-fit que chama sempre o último recipiente), apenas atualizamos um valor, ganhando desempenho.

Nos métodos *offline* ordenamos os tamanhos em ordem não crescente usando a função *sort* do pyhton e depois chamamos a heurística do método *online* correspondente passando como argumento a lista ordenada. Para evitar problemas na recuperação dos nomes dos arquivos trabalhamos com uma cópia da lista original. Mais detalhes da implementação específios de cada heurística podem ser encontrados nos comentários do código-fonte em A.2

5 Testes e Resultados Obtidos

Para testar o programa fizemos um script em Python **gera.py**, apresentado em A.3, que recebe os seguintes parâmetros:

- -l ou --lista: Numero maximo de elementos um um recipiente
- -c ou --capacidade: Capacidade da midia
- -q ou --quantidade: Quantidade de recipientes

Além dos parâmetros de impressão e tipo de heurística que foram apresentados para o *metodos.py*.

Este script gera uma lista de itens de tamanhos aleatórios seguindo as restrições passadas na sua chamada e já faz a chamada das heurísticas com os parâmetros correspondentes. Os recipientes inicialmente são sempre gerados completamente cheios

Variamos a quantidade de recipientes (que será o valor ótimo), capacidade e número máximo de itens por recipiente e apresentamos a média do tempo e resultado obtido (arredondando para cima) em 8 execuções por cada uma das heurísticas nas tabelas 1 e 2. Para as capacidades, optamos pelas capacidades médias de um cd e de um dvd, em bytes e variamos os tamanhos de modo a simular mídias com mais ou menos arquivos, mas não usando quantidades muito elevadas de arquivos pois estamos interessados, nesse caso, em arquivos grandes.

Devido ao valor ótimo não ter sido alcançado nenhuma vez, fizemos uma segunda leva de testes, também calculando a média de 8 execuções de cada configuração, mas dessa fez retirando alguns elementos da lista de itens, mas sem ultrapassar um número inteiro sorteado entre 1 e a capacidade da mídia. Dessa forma o ótimo continua o mesmo que o anterior para cada caso, mais há uma folga maior para o método trabalhar, e também consideramos que este tipo de caso se aproxima mais da realidade. Esses resultados estão mostrados nas tabelas 3 e 4.

O valor do tempo em segundos consumido por cada heurística aparece entre parênteses.

Ótimo	N. Itens	NF	FF	BF	WF	AWF	BFD	FFD
10	10	12 (0)	11 (0)	11 (0)	11 (0)	11 (0)	11 (0)	11 (0)
10	100	11 (0)	11 (0)	11 (0)	11 (0)	11 (0,01)	11 (0)	11 (0)
10	500	11 (0,01)	11 (0,01)	11 (0,01)	11 (0,01)	11 (0,01)	11 (0,02)	11 (0,01)
10	1000	11 (0,03)	11 (0,02)	11 (0,02)	11 (0,02)	11 (0,03)	11 (0,03)	11 (0,03)
100	10	119 (0)	101 (0,01)	101 (0,01)	106 (0,02)	105 (0,01)	101 (0,02)	101 (0,02)
100	100	104 (0,01)	101 (0,1)	101 (0,11)	101 (0,13)	101 (0,15)	101 (0,15)	101 (0,15)
100	500	102 (0,08)	101 (0,49)	101 (0,5)	101 (0,62)	101 (0,65)	101 (0,76)	101 (0,72)
100	1000	102 (0,31)	101 (1)	101 (1,04)	101 (1,24)	101 (1,37)	101 (1,54)	101 (1,46)
1000	10	1183 (0)	1005 (1)	1005 (1,02)	1055 (1,44)	1048 (1,5)	1001 (1,52)	1001 (1,46)
1000	100	1036 (0,07)	1001 (9,04)	1001 (9,33)	1007 (12,96)	1006 (13,64)	1001 (14,22)	1001 (13,58)
1000	500	1009 (0,88)	1001 (45,14)	1001 (48,27)	1002 (64,93)	1002 (69,13)	1001 (73,97)	1001 (67,29)
1000	1000	1006 (3,03)	1001 (88,94)	1001 (98,74)	1001 (128,35)	1001 (133,88)	1001 (146,57)	1001 (133,35)

Tabela 1: Resultados dos testes para um cd (737280000 Bytes) - sem folga

Ótimo	N. Itens	NF	FF	BF	WF	AWF	BFD	FFD
10	10	13 (0)	11 (0)	11 (0)	11 (0)	12 (0)	11 (0)	11 (0)
10	100	11 (0)	11 (0)	11 (0)	11 (0,01)	11 (0)	11 (0)	11 (0)
10	500	11 (0,05)	11 (0,02)	11 (0,01)	11 (0,02)	11 (0,02)	11 (0,02)	11 (0,02)
10	1000	11 (0,26)	11 (0,03)	11 (0,03)	11 (0,03)	11 (0,04)	11 (0,04)	11 (0,05)
100	10	117 (0)	101 (0,02)	101 (0,02)	106 (0,02)	105 (0,02)	101 (0,02)	101 (0,02)
100	100	105 (0,03)	101 (0,16)	101 (0,15)	101 (0,18)	101 (0,19)	101 (0,23)	101 (0,24)
100	500	102 (0,61)	101 (0,82)	101 (0,8)	101 (0,96)	101 (1,01)	101 (1,2)	101 (1,2)
100	1000	102 (2,49)	101 (1,68)	101 (1,67)	101 (1,95)	101 (2,07)	101 (2,44)	101 (2,47)
1000	10	1182 (0,01)	1004 (1,39)	1005 (1,35)	1054 (1,86)	1048 (1,97)	1001 (2,02)	1001 (2,07)
1000	100	1034 (0,31)	1001 (15,12)	1001 (15,2)	1007 (19,21)	1006 (20,43)	1001 (23,26)	1001 (23,09)
1000	500	1010 (6,22)	1001 (77,59)	1001 (78,17)	1002 (96,9)	1002 (99,9)	1001 (116,1)	1001 (114,28)
1000	1000	1006 (23,96)	1001 (154,13)	1001 (156,28)	1001 (190,49)	1001 (197,27)	1001 (232,38)	1001 (229,07)

Tabela 2: Resultados dos testes para um dvd (47000000000 Bytes) - sem folga

Ótimo	N. Itens	NF	FF	BF	WF	AWF	BFD	FFD
10	10	12 (0)	11 (0)	11 (0)	11 (0)	11 (0)	10 (0)	10 (0)
10	100	11 (0)	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)
10	500	10 (0,01)	10 (0,01)	10 (0,02)	10 (0,01)	10 (0,01)	10 (0,01)	10 (0,02)
10	1000	10 (0,03)	10 (0,02)	10 (0,02)	10 (0,02)	10 (0,03)	10 (0,02)	10 (0,03)
100	10	118 (0)	101 (0,02)	101 (0,01)	106 (0,01)	106 (0,02)	101 (0,02)	101 (0,02)
100	100	103 (0,01)	100 (0,1)	100 (0,1)	101 (0,13)	101 (0,14)	100 (0,15)	100 (0,15)
100	500	101 (0,09)	100 (0,51)	100 (0,52)	100 (0,63)	100 (0,67)	100 (0,78)	100 (0,74)
100	1000	101 (0,29)	100 (0,98)	100 (1,02)	100 (1,25)	100 (1,33)	100 (1,51)	100 (1,44)
1000	10	1181 (0)	1004 (1,02)	1005 (1,03)	1054 (1,41)	1049 (1,46)	1001 (1,54)	1001 (1,48)
1000	100	1032 (0,08)	1001 (9,2)	1001 (9,47)	1007 (13,23)	1006 (13,77)	1001 (14,41)	1001 (13,77)
1000	500	1009 (0,9)	1000 (45,49)	1000 (48,5)	1002 (64,01)	1001 (67,74)	1000 (74,54)	1000 (67,86)
1000	1000	1005 (3,13)	1000 (90,69)	1000 (101,03)	1001 (127,74)	1001 (138,48)	1000 (150,15)	1000 (136,62)

Tabela 3: Resultados dos testes para um cd (737280000 Bytes) - com folga

Ótimo	N. Itens	NF	FF	BF	WF	AWF	BFD	FFD
10	10	12 (0)	11 (0)	11 (0)	11 (0)	11 (0)	11 (0)	11 (0)
10	100	11 (0,01)	10 (0)	10 (0)	11 (0)	11 (0,01)	10 (0)	10 (0)
10	500	10 (0,07)	10 (0,01)	10 (0,02)	10 (0,01)	10 (0,02)	10 (0,02)	10 (0,02)
10	1000	10 (0,24)	10 (0,03)	10 (0,03)	10 (0,03)	10 (0,03)	10 (0,04)	10 (0,04)
100	10	118 (0)	101 (0,02)	101 (0,01)	106 (0,02)	105 (0,02)	101 (0,02)	101 (0,02)
100	100	104 (0,03)	100 (0,16)	100 (0,15)	101 (0,19)	101 (0,2)	100 (0,23)	100 (0,23)
100	500	102 (0,62)	100 (0,83)	100 (0,81)	101 (0,95)	100 (1,01)	100 (1,21)	100 (1,22)
100	1000	101 (2,32)	100 (1,62)	100 (1,6)	101 (1,87)	101 (1,98)	100 (2,37)	100 (2,39)
1000	10	1181 (0,01)	1004 (1,41)	1004 (1,36)	1053 (1,9)	1048 (2)	1000 (2,03)	1000 (2,1)
1000	100	1033 (0,33)	1001 (15,51)	1001 (15,58)	1007 (19,78)	1006 (20,71)	1001 (23,75)	1001 (23,57)
1000	500	1009 (6,38)	1000 (78,97)	1000 (79,67)	1002 (95,83)	1002 (101,95)	1000 (118,14)	1000 (116,37)
1000	1000	1005 (24,36)	1000 (155,52)	1000 (158,77)	1001 (195,04)	1001 (200,74)	1000 (234,86)	1000 (231,44)

Tabela 4: Resultados dos testes para um dvd (47000000000 Bytes) - com folga

Observando os resultados encontrados vemos que entradas com maior quantidade de arquivos de menor tamanho têm um custo computacional mais elevado (inerente ao tamanho da lista de entrada), mas por outro lado levam à resultados mais precisos.

Listas com itens de tamanho maior são avaliadas mais rapidamente, porém os resultados são mais distantes do valor ótimo, mas ainda assim não se aproximaram dos limites de pior caso.

O valor ótimo exato não foi atingido, mas frequentemente foi alcançado com uma unidade a mais no caso das mídias completamente cheias. Reduzindo da soma total dos itens uma quantidade menor do que um dvd (ou cd) obtivemos o valor ótimo com uma grande frequência. É importante notar que como os valores foram arredondados para cima as heuristicas que chegaram no ótimo na tabela encontraram o valor ótimo nas 8 vezes.

Conforme já era esperado, o Next-Fit foi executado mais rapidamente, e teve os piores resultados em relação ao valor ótimo. O Worst-Fit também mostrou resultados mais distantes do valor ótimo. Ainda assim, mantiveram-se afastados do limite de pior caso. As heurísticas *offline* tiveram os melhores resultados e um custo de tempo um pouco maior que suas correspondentes *online*.

O tamanho da mídia exerceu pouca influência no valor ótimo, por outro lado, computar o empacotamento em dvds levou mais tempo que em cds.

6 Conclusão

De uma maneira geral, o uso de heurísticas para resolver o *Bin-packing*, ou mais especificamente, o armazenamento de arquivos grandes em dvds, foi satisfatório, em particular as heurísticas *offline*.

Apesar do tamanho maior do arquivo fazer com que o algoritmo tenha mais dificuldade em atingir o ótimo exato, nas heurísticas *offline* mesmo para o caso de 1000 dvds o ótimo passou apenas em uma unidade, no pior caso, sem um aumento computacional que invibilizasse sua execução.

Como trabalhos futuros pode ser implementado algum método exato de resolução do *Bin-packing* que pode ser encontrado por exemplo em [7], ou métodos meta-heurísticos.

A Códigos-Fonte

A.1 organizador.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import wx
import wx.aui
import metodos
import time
import os
import stat, sys

# Tranforma um valor em bytes em megabytes
def Bytes2Mega(num):
    return round(num/float(1024*1024),3)

# Recebe uma string e verifica se o número é válido
def validate(number, decimal = '.'):
    ponto = False
    if number =="":
        return False
    for a in number:
        if not a.isdigit():
            if a == decimal and not ponto:
                ponto = True
            else:
                return False
    return True

# Classe com a interface
class Organizador(wx.Frame):
    def __init__(self, *args, **kwds):

        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)
        self.SetSize(wx.Size(420,530))
```

```
# Associa cada opção da radio box com uma
    função
self.metodos = {0: metodos.NF,
                1: metodos.BF,
                2: metodos.FF,
                3: metodos.WF,
                4: metodos.AWF,
                5: metodos.BFD,
                6: metodos.FFD}


# Vão receber os tamanhos e os nomes dos
    arquivos selecionados
self.tamanhos = []
self.itens = []

# Painel superior da esquerda
self.painel_superior_esquerda = wx.Panel(self,
    -1) # Painel superior da esquerda
self.metodo = wx.RadioBox(self.
    painel_superior_esquerda, -1, u"Método
    Utilizado:", choices=["Next Fit (NF)", "Best
     Fit (BF)", "First Fit (FF)", "Worst Fit (WF
    )", "Almost Worst Fit (AWF)", "Best Fit
    Decreasing (BFD)", "First Fit Decreasing (
    FFD)"], majorDimension=0, style=wx.
    RA_SPECIFY_ROWS)
self.tamanho_text = wx.TextCtrl(self.
    painel_superior_esquerda, -1, "")
self.iniciar = wx.Button(self.
    painel_superior_esquerda, -1, "Iniciar")

# Painel superior da direita
self.painel_superior_direita = wx.Panel(self,
    -1) # Painel superior da direita
self.arquivos = wx.ListCtrl(self.
    painel_superior_direita, -1, style=wx.
```

```
        LC_REPORT|wx.SUNKEN_BORDER| wx.LC_SINGLE_SEL
        ) # Lista com os arquivos a serem alocados
    self.adicionar =  wx.Button(self.
        painel_superior_direita, -1,"Adicionar") #
        Botão para acrescentar arquivos na lista
    self.remover = wx.Button(self.
        painel_superior_direita, -1, "Remover") #
        Botão para remover arquivos da lista

    # Títulos das colunas
    self.arquivos.InsertColumn(0, "Nome")
    self.arquivos.InsertColumn(1, "Tamanho (Mbytes)
        ")

    #Painel inferior
    self.painel_inferior = wx.Panel(self, -1) #
        Painel inferior
    self.saida = wx.TextCtrl(self.painel_inferior,
        -1, "", style=wx.TE_MULTILINE|wx.TE_READONLY
        )
    self.valor_tempo = wx.TextCtrl(self.
        painel_inferior, -1, "", style=wx.
       TE_READONLY)
    self.valor_dvd = wx.TextCtrl(self.
        painel_inferior, -1, "", style=wx.
       TE_READONLY)

    # Seleção inicial
    self.metodo.SetSelection(0)

    # Sizer do painel superior da esquerda
    sizer_5 = wx.BoxSizer(wx.VERTICAL)
    sizer_5.Add(self.metodo, 1, wx.EXPAND, 1)
    sizer_5.Add( wx.StaticText(self.
        painel_superior_esquerda, -1, u"Tamanho da
        Mídia (Mb):"), 0, 0, 0)
    sizer_5.Add(self.tamanho_text, 0,  wx.EXPAND,
        1)
```

```
sizer_5.Add( wx.StaticText(self.
    painel_superior_esquerda, -1, u"(1 Kilobyte
    = 1024 Bytes)"), 0, 0, 0)
sizer_5.Add(self.iniciar, 0, wx.EXPAND|wx.
    ALIGN_BOTTOM, 1)
self.painel_superior_esquerda.SetSizer(sizer_5)

# Sizers do painel superior da direita
sizer_4 = wx.BoxSizer(wx.HORIZONTAL)
sizer_4.Add(self.adicionar, 1, wx.EXPAND, 1)
sizer_4.Add(self.remover, 1, wx.EXPAND, 1)
sizer_3 = wx.BoxSizer(wx.VERTICAL)
sizer_3.Add(self.arquivos, 1, wx.EXPAND, 1)
sizer_3.Add(sizer_4, 0, wx.EXPAND, 1)
self.painel_superior_direita.SetSizer(sizer_3)

# Sizers do painel inferior
sizer_6 = wx.StaticBoxSizer(wx.StaticBox(self.
    painel_inferior, -1, u"Sugestão de Gravação"
    ), wx.VERTICAL)
sizer_6.Add(self.saida, 1, wx.EXPAND, 1)
sizer_7 = wx.BoxSizer(wx.HORIZONTAL)
sizer_7.Add( wx.StaticText(self.painel_inferior
    , -1, "Tempo Gasto:"), 0, 0, 0)
sizer_7.Add(self.valor_tempo, 0, 0, 0)
sizer_7.Add((20, 20), 1, 0, 0)
sizer_7.Add(wx.StaticText(self.painel_inferior,
     -1, u"Nº de DVDs:"), 0, 0, 0)
sizer_7.Add(self.valor_dvd, 0, 0, 0)
sizer_6.Add(sizer_7, 0, wx.EXPAND, 1)
self.painel_inferior.SetSizer(sizer_6)

# Cria o gerenciador de paineis e adiciona os
    paineis a ele
self.mgr = wx.aui.AuiManager()
self.mgr.SetManagedWindow(self)
self.mgr.AddPane(self.painel_superior_direita,
    wx.aui.AuiPaneInfo().Center().Floatable(
```

```
            False).CloseButton(False).Name("Arquivos").
                Caption(u"Arquivos").BestSize((200, 200)).
                Movable(False).MinSize((200, 200)))
        self.mgr.AddPane(self.painel_superior_esquerda,
            wx.aui.AuiPaneInfo().Left().BestSize((200,
            200)).MinSize((200, 200)).Floatable(False).
            Caption(u"Opções").CloseButton(False).
            Movable(False).Name("Opcoes"))
        self.mgr.AddPane(self.painel_inferior ,wx.aui.
            AuiPaneInfo().Bottom().BestSize((200, 200)).
            MinSize((200, 200)).Floatable(False).Caption
            (u"Resultado").CloseButton(False).Movable(
            False).Name("Resultado"))
        self.mgr.Update()

        # Conecta as widgets às funções
        self.Bind(wx.EVT_BUTTON, self.organiza, self.
            iniciar)
        self.Bind(wx.EVT_BUTTON, self.adicionar_arq,
            self.adicionar)
        self.Bind(wx.EVT_BUTTON, self.remover_arq, self
            .remover)
        self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.
            selecionouItem, self.arquivos)

    # Faz a sugestão de gravação dos arquivos passados
        pelo método selecionado
    def organiza(self, evt):
        # descobre o método
        num = self.metodo.GetSelection()
        metodo = self.metodos[num]

        # Se o tamanho inserido pelo usuário for válido
            usa esse, senão usa o padrão de um dvd
        tamanho = self.tamanho_text.GetValue()
        if validate(tamanho):
            tamanho = int(float(tamanho) *
                1024.0*1024.0)
```

```
        else:
            tamanho = 4700000000

        # Chama o método selecionado, contando o tempo
        inicio = time.time()
        bins = metodo(self.tamanhos, tamanho)
        fim = time.time()

        # Processa a saída obtida
        numero = len(bins)
        self.valor_dvd.SetValue(str(numero))
        self.imprime(bins)
        self.valor_tempo.SetValue(str(round(fim -
            inicio, 4)))

    # Adiciona arquivos à lista de arquivos
    def adicionar_arq(self, evt):
        # Cria a caixa de seleção de arquivos
        arq = wx.FileDialog(self, message="Escolha o(s)
            arquivo(s)", defaultDir=os.getcwd(),
            defaultFile="", wildcard="Todos arquivos
            (*.*)|*.*", style=wx.OPEN | wx.MULTIPLE | wx
            .CHANGE_DIR| wx.FD_FILE_MUST_EXIST)

        # Só continua se o usuário clicou em ok
        if arq.ShowModal() == wx.ID_OK:
            paths = arq.GetPaths()      # Arquivos
                selecionados
            barra = os.path.sep         # Para não dar
                problema de sistema operacional

            # Vai percorrer cada item da lista de
                arquivos para adicionar na tela
            for item in paths:
                self.itens.append(item.split(barra)
                    [-1])
                self.tamanhos.append((os.stat(item)[
                    stat.ST_SIZE]))
```

```
                ind = self.arquivos.InsertStringItem(
                    sys.maxint, self.itens[-1])
                self.arquivos.SetStringItem(ind, 1, str
                    (Bytes2Mega(self.tamanhos[-1])))
                self.arquivos.SetItemData(ind,len(self.
                    itens)-1) # Para manter uma relação
                    entre os itens e a lsita
            self.arquivos.SetColumnWidth(0, wx.
                LIST_AUTOSIZE)
        arq.Destroy()

    # Remove arquivos da lista de arquivos e das listas
        com os itens e tamanhos
    def remover_arq(self, evt):
        indice = self.arquivos.GetItemData(self.
            itemAtual)
        self.arquivos.DeleteItem(self.itemAtual)
        self.itens.pop(indice)
        self.tamanhos.pop(indice)

    # Trata o item selecionado (para saber quem remover
        quando o botão for clicado)
    def selecionouItem(self, event):
        self.itemAtual = event.m_itemIndex

    # Imprime o resultado do algoritmo na tela
    def imprime(self, bins):
        self.saida.SetValue("") # Limpa a saída
        self.arquivos.DeleteAllItems() # Limpa a lista
        # Percorre cada um dos recipientes
        for dvd in range(len(bins)):
            self.saida.AppendText("DVD: " + str(dvd +
                1) + " (" + str(Bytes2Mega(sum(bins[dvd
                ]))) + " Mb ocupados):\n")
            #Percorre cada item de um recipiente
            for arquivo in bins[dvd]:
                ind = self.tamanhos.index(arquivo)
                arq = self.itens.pop(ind)
```

```
                    tam = str(Bytes2Mega(self.tamanhos.pop(
                        ind)))
                    self.saida.AppendText(unicode(arq) + "
                        - " + tam +" Mbytes \n")
                self.saida.AppendText("\n")

if __name__ == "__main__":
    app = wx.PySimpleApp(0)
    wx.InitAllImageHandlers()
    janela = Organizador(None, -1, "Organizador de
        arquivos em DVDs")
    app.SetTopWindow(janela)
    janela.Show()
    app.MainLoop()
```

A.2 metodos.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from optparse import OptionParser
import time

#Faz a alocação utilizando Next-Fit
def NF(itens, capacidade):
    recipientes = [[]] # Começa com um recipiente vazio
    #Percorre os itens
    for a_i in itens:
        # Verifica se dá para alocar a_i no recipiente
            aberto e abre um novo caso não dê
        if a_i <= (capacidade - sum(recipientes[-1])):
            recipientes[-1].append(a_i)
        else:
            recipientes.append([a_i]) # Abre um
                recipiente novo com a_i alocado nele
    return recipientes

#Faz a alocação utilizando First-Fit
```

```
def FF(itens, capacidade):
    recipientes = [[]] # Começa com um recipiente vazio
    niveis = [0] # Nivel do recipiente vazio é zero
    #Percorre os itens
    for a_i in itens:
        alocou = False
        # Percorre os recipientes
        for b_j, nivel in zip(recipientes, range(len(
            niveis))):
            # Verifica se dá para alocar a_i
            if a_i <= (capacidade - niveis[nivel]):
                b_j.append(a_i)
                niveis[nivel] += a_i
                alocou = True
                break
        # Abre um recipiente novo e atualiza seu nivel
            se a_i não coube em nenhum dos abertos
        if not alocou:
            recipientes.append([a_i])
            niveis.append(a_i)
    return recipientes

#Faz a alocação utilizando Best-Fit
def BF(itens, capacidade):
    recipientes = [[]] # Começa com um recipiente vazio
    niveis = [0] # Nivel do recipiente vazio é zero
    #Percorre os itens
    for a_i in itens:
        melhor_ind = -1
        melhor_qtde = 0
        # Percorre os recipientes guardando o que
            possui maior espaço usado após alocação
        for b_j, nivel in zip(recipientes, niveis):
            # Verifica se dá para alocar a_i
            if a_i <= (capacidade - nivel):
                # Atualiza o melhor valor, se
                    necessário
                if nivel > melhor_qtde: # Desigualdade
```

```
                    estrita para desempatar pelo menor
                    índice
                        melhor_ind = recipientes.index(b_j)
                        melhor_qtde = nivel
        if melhor_ind is not -1: # Conseguiu um
            recipiente bom
            recipientes[melhor_ind].append(a_i)
            niveis[melhor_ind] += a_i
        else:
            recipientes.append([a_i]) # Abre um
                recipiente novo com a_i alocado nele
            niveis.append(a_i)
    recipientes.remove([])
    return recipientes

#Faz a alocação utilizando Worst-Fit
def WF(itens, capacidade):
    recipientes = [[]] # Começa com um recipiente vazio
    niveis = [0] # Nivel do recipiente vazio é zero
    #Percorre os itens
    for a_i in itens:
        melhor_qtde = capacidade + 1
        melhor_ind = -1
        # Percorre os recipientes guardando o que
            possui menor espaço usado após alocação
        for b_j, nivel in zip(recipientes, niveis):
            # Verifica se dá para alocar a_i
            if a_i <= (capacidade - nivel):
                # Atualiza o melhor valor, se
                    necessário
                if melhor_qtde > nivel: # Desigualdade
                    estrita para desempatar pelo menor
                    índice
                        melhor_ind = recipientes.index(b_j)
                        melhor_qtde = nivel
        if melhor_ind is not -1: # Conseguiu um
            recipiente bom
            recipientes[melhor_ind].append(a_i)
```

```
            niveis[melhor_ind] += a_i
        else:
            recipientes.append([a_i]) # Abre um
                recipiente novo com a_i alocado nele
            niveis.append(a_i)
    return recipientes

#Faz a alocação utilizando Almost-Worst-Fit
def AWF(itens, capacidade):
    recipientes = [[]] # Começa com um recipiente vazio
    niveis = [0] # Nivel do recipiente vazio é zero
    #Percorre os itens
    for a_i in itens:
        segundo_melhor_qtde = capacidade + 2
        melhor_qtde = capacidade + 1
        melhor_ind = -1
        melhor_ind_2 = -1
        # Percorre os recipientes guardando os que
            possuem menor e o segundo menor espaço usado
            após alocação
        for b_j, nivel in zip(recipientes, niveis):
            # Verifica se dá para alocar a_i
            if a_i <= (capacidade - nivel):
                # Atualiza o melhor e o segundo melhor
                    valor, se necessário
                # Se cabe no com menos espaço, cabe no
                    segundo com menos espaço
                if melhor_qtde > nivel: # Desigualdade
                    estrita para desempatar pelo menor
                    índice
                    segundo_melhor_qtde = melhor_qtde
                    melhor_qtde = nivel
                    if melhor_ind is not -1: # Caso o
                        melhor ainda não tenha sido
                        atualizado
                        melhor_ind_2 = melhor_ind
                        melhor_ind = recipientes.index(
                            b_j)
```

```
                    else:
                        melhor_ind_2 = recipientes.
                            index(b_j)
                elif segundo_melhor_qtde > nivel:
                    segundo_melhor_qtde = nivel
                    melhor_ind_2 = recipientes.index(
                        b_j)
        if melhor_ind_2 is not -1: # Achou
            recipientes[melhor_ind_2].append(a_i)
            niveis[melhor_ind_2] += a_i
        else:
            recipientes.append([a_i]) # Abre um
                recipiente novo com a_i alocado nele
            niveis.append(a_i)
    return recipientes

#Faz a alocação utilizando Best-Fit-Decreasing
def BFD(itens, capacidade):
    itens_ord = [i for i in itens] # copia pois precisa
        preservar a ordem do original
    itens_ord.sort(reverse = True)
    recipientes = BF(itens_ord, capacidade)
    return recipientes

#Faz a alocação utilizando First-Fit-Decreasing
def FFD(itens, capacidade):
    itens_ord = [i for i in itens] # copia pois precisa
        preservar a ordem do original
    itens_ord.sort(reverse = True)
    recipientes = FF(itens_ord, capacidade)
    return recipientes

# Imprime o resultado do algoritmo na tela
def imprime(bins, tempo, imprime_dentro, imprime_fora):
    print "  Tempo total: " + str(tempo) + " segundos",
        "  Quantidade de Dvds: " + str(len(bins))
    # Percorre cada um dos recipientes
    for dvd in range(len(bins)):
```

```
        if imprime_fora:
            print "  DVD " + str(dvd + 1)  + " (" + str
               (sum(bins[dvd])) + " bytes ocupados)"
            #Percorre cada item de um recipiente
            if imprime_dentro:
                for arquivo in bins[dvd]:
                    print "     " + str(arquivo)
                print

# Trata caso o programa tenha sido chamado fora da
    interface gráfica
def main():

    metodos_opt = {'nf': False, 'bf': False, 'ff':
       False, 'wf': False,'awf': False, 'ffd': False,'
       bfd': False}
    metodos = {'nf': NF, 'bf': BF, 'ff': FF, 'wf': WF,'
       awf': AWF, 'ffd': FFD,'bfd': BFD}

        # Cria e configura um parser para tratar as
           opções de entrada passadas pelo usuário
    parser = OptionParser()

        # Parametros da função
    parser.add_option("-l", "--lista", help=u"Arquivo
       com os dados de entrada (capacidade e itens)")

    #Imprime os objetos alocados
    parser.add_option("-i", "--imprime",          help=u"
       Imprime os elementos de cada dvd", action="
       store_true", default=False)
    parser.add_option("-j", "--imprime2",         help=u"
       Imprime o nivel de cada dvd", action="store_true
       ", default=False)

        # Tipo de empacotamento
    parser.add_option("-n", "--nf",       help=u"Faz o
       empacotamento utilizando next-fit.", action="
```

```
    store_true", default=False)
parser.add_option("-b", "--bf",      help=u"Faz o
    empacotamento utilizando best-fit.", action="
    store_true", default=False)
parser.add_option("-f", "--ff",      help=u"Faz o
    empacotamento utilizando first-fit.", action="
    store_true", default=False)
parser.add_option("-w", "--wf",      help=u"Faz o
    empacotamento utilizando worst-fit.", action="
    store_true", default=False)
parser.add_option("-a", "--awf",     help=u"Faz o
    empacotamento utilizando almost-worst-fit.",
    action="store_true", default=False)
parser.add_option("-d", "--ffd",     help=u"Faz o
    empacotamento utilizando first-fit-decreasing.",
     action="store_true", default=False)
parser.add_option("-g", "--bfd",     help=u"Faz o
    empacotamento utilizando best-fit-decreasing.",
    action="store_true", default=False)

(options, args) = parser.parse_args()

    # Trata os tamanhos dos arquivos
if options.lista:
    arquivo = file(options.lista, "r")
    capacidade = int(arquivo.readline())
    lista_aux = arquivo.readline().split(' ')
    arquivo.close()
    lista = []
    for item in lista_aux:
        # Ignora os tamanhos maiores que a
            capacidade da mídia
        if int(item) > capacidade:
            print "O elemento " + str(item) + u"
                não será utilizado por execeder a
                capacidade " + str(capacidade)
        else:
            lista.append(int(item))
```

```
else:
    print "Por favor, especifique o arquivo de
       entrada"
    exit()

impressao = options.imprime
impressao2 = options.imprime2

# Trata a escolha dos métodos
if options.nf:
    metodos_opt['nf'] = True

if options.bf:
    metodos_opt['bf'] = True

if options.ff:
    metodos_opt['ff'] = True

if options.wf:
    metodos_opt['wf'] = True

if options.awf:
    metodos_opt['awf'] = True

if options.ffd:
    metodos_opt['ffd'] = True

if options.bfd:
    metodos_opt['bfd'] = True

#chama cada um dos métodos escolhidos e imprime o
   resultado obtido
metodos_lista = ['nf', 'bf','ff', 'wf','awf', 'ffd'
   ,'bfd']
#for chave, valor in metodos_opt.iteritems():
for valor in metodos_lista:
    if metodos_opt[valor]:
        inicio = time.time()
```

```
            bins = metodos[valor](lista, capacidade)
            #bins = metodos[chave](lista, capacidade)
            fim = time.time()
            # Processa a saída obtida
            print u"Metodo " + valor.upper(),
            imprime(bins, fim - inicio, impressao,
                impressao2)

    # fim!
    return(0)

if __name__ == "__main__":
    main()
```

A.3 gera.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-


import random

import sys

import subprocess

import math

from optparse import OptionParser


# Gera uma lista ateatória de elementos

def lista_ale(numero, capacidade):
    numero_novo = random.randint(1, numero)
```

```
    a = [random.random() for i in range(numero_novo)]
    t = capacidade/sum(a)
    b = [math.ceil(a[i]*t) for i in range(numero_novo)]
    diferenca = capacidade-sum(b)
    if diferenca > 0:
        b.append(diferenca)
    else:
        for i in b:
            item = b.pop()
            t = sum(b)
            if t < capacidade:
                b.append(capacidade-t)
                break
            elif t == capacidade:
                break
    return map(int,b)


# Junta as listas em uma string unica

def junta(a, capacidade):
    # Junta as listas

    lista = []

    for i in a:

        lista.extend(i)
    # Sorteia uma ordem nova
    random.shuffle(lista)



    #Remove alguns itens sem passar a capacidade, ou
       seja, sem alterar o ótimo

    tamanho = random.randint(1, int(capacidade))
```

```
    continua = True

    while lista[-1] <= tamanho:

        dif = lista.pop()

        tamanho -= dif


    # Passa para string

    s = ''
    for i in lista:
        s += str(i) + ' '

    return s.rstrip(' ') # Tira o ultimo espaço


# Grava as listas em um arquivo de entrada

def listas2arq(listas, capacidade, nome):
    lista_file = file(nome, "w")
    lista_file.write(capacidade + '\n')
    lista_file.write(junta(listas, capacidade))
    lista_file.close()




# Função principal

def main():



        # Cria e configura um parser para tratar as
           opções de entrada passadas pelo usuário
```

```
parser = OptionParser()


    # Parametros da função

parser.add_option("-l", "--lista",  help=u"Numero
   maximo de elementos da lista")

parser.add_option("-q", "--quantidade",      help=u"
   Numero de recipientes")

parser.add_option("-c", "--capacidade",      help=u"
   Capacidade da midia (em Bytes)")

parser.add_option("-i", "--imprime",         help=u"
   Imprime os elementos de cada dvd", action="
   store_true", default=False)
parser.add_option("-j", "--imprime2",        help=u"
   Imprime o nivel de cada dvd", action="store_true
   ", default=False)



    # Tipo de empacotamento

parser.add_option("-n", "--nf",      help=u"Faz o
   empacotamento utilizando next-fit.", action="
   store_true", default=False)

parser.add_option("-b", "--bf",      help=u"Faz o
   empacotamento utilizando best-fit.", action="
   store_true", default=False)

parser.add_option("-f", "--ff",      help=u"Faz o
   empacotamento utilizando first-fit.", action="
   store_true", default=False)
```

```
parser.add_option("-w", "--wf",      help=u"Faz o
    empacotamento utilizando worst-fit.", action="
    store_true", default=False)

parser.add_option("-a", "--awf",     help=u"Faz o
    empacotamento utilizando almost-worst-fit.",
    action="store_true", default=False)

parser.add_option("-d", "--ffd",     help=u"Faz o
    empacotamento utilizando first-fit-decreasing.",
     action="store_true", default=False)

parser.add_option("-g", "--bfd",     help=u"Faz o
    empacotamento utilizando best-fit-decreasing.",
    action="store_true", default=False)



(options, args) = parser.parse_args()


print "Capacidade: ", options.capacidade, "
    Quantidade ", options.quantidade, " Elementos: "
    , options.lista


    # Trata a capacidade da mídia

if options.capacidade:

    capacidade = int(options.capacidade)

else:

    print "Por favor, especifique a capacidade"

    exit()
```

```
# Trata o numero de elementos

if options.lista:

    numero = int(options.lista)

else:

    print "Por favor, especifique o numero maximo
       de elementos"

    exit()


# Trata o numero de elementos

if options.quantidade:

    quantidade = int(options.quantidade)

else:

    print "Por favor, especifique a quantidade de
       recipientes"

    exit()


bins = []

for b in range(quantidade):

    bins.append(lista_ale(numero, capacidade))
```

```
metodos = ""
nome = 'entrada.txt'

listas2arq(bins, options.capacidade, nome)
itens = " -l\"" + nome + "\""


if options.imprime:

    impressao = " -i"

else:

    impressao = ""

if options.imprime2:

    impressao += " -j"


# Trata a escolha dos métodos

if options.nf:

    metodos += " -n"


if options.bf:

    metodos += " -b"
```

```
if options.ff:
    metodos += " -f"

if options.wf:
    metodos += " -w"

if options.awf:
    metodos += " -a"

if options.ffd:
    metodos += " -d"

if options.bfd:
    metodos += " -g"

if sys.platform == "win32":
    params = "metodos"+ext + itens + metodos +
        impressao
else:
```

```
        params = "./metodos" + ext + itens + metodos +
            impressao


    p = subprocess.Popen(params, shell=True)

    p.wait()


    # fim!

    return(0)


if __name__ == "__main__":

    # Para tentar gerar executavel depois...

    global ext

    namep = str(sys.argv[0])

    if namep.endswith('.py'):

        ext = ".py"

    elif sys.platform == "win32":

        ext = ".exe"

    else:

        ext = ""

    main()
```

Referências

[1] D. S. Johnson: *Near-optimal bin packing algorithms*, Massachusetts Institute of Technology (1973)

[2] E. C. Xavier, F. K. Miyazawa: *Algoritmos para Problemas de Empacotamento*, Anais do XXVII Congresso da SBC, CTD, XX Concurso de Teses e Dissertações, p1966 - 1973, Rio de Janeiro (2007)

[3] D. Hochbaum: *Approximation algoritms for NP-hard problems*, PWS Publishing Company, Boston (1995)

[4] F. Wronski: *Alocação Dinâmica de Tarefas em NoCs Malha com Redução do Consumo de Energia*, Universidade Federal do Rio Grande do Sul, Dissertação de Mestrado, Porto Alegre, (2007)

[5] F. C: Rodrigues: *Programação Dinâmica Eficiente com Algoritmos Cache-Oblivious*, Universidade Federal do Rio Grande do Sul, Dissertação de Mestrado, Porta Alegre, (2008)

[6] E. G. Coffman et. Al: *Average Analysis of on-line Bin-packing Algorithms*, OR-Seminar, Michaelmas (1996)

[7] S. Martello, P. Toth: *Knapsack Problems - Algorithms and Computer Implementations*, John Wilew & Sons (1990)

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: *Introduction to Algorithms*, 2nd edition, MIT Press & McGraw-Hill, (2001).