Trabalho Prático 2
Backtracking e Branch-and-Bound

Vitor R. Ricoy[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Caixa Postal 486 – 31.270-901 – Belo Horizonte – MG – Brazil

vitorricoy@ufmg.br

Resumo. *Este artigo descreve a implementação e os resultados do Trabalho Prático 2 da disciplina Algoritmos II. Foi proposta a implementação de dois algoritmos que resolvam de forma ótima o problema da mochila binário: um algoritmo de backtrack e um algoritmo de Branch-and-Bound. Ambos foram implementados e executados para os arquivos de testes fornecidos.*

1. Introdução

Para esse trabalho foram implementados dois algoritmos que resolvem o problema da mochila binário. O primeiro deles é uma algoritmo de Backtracking que analisa extensivamente as soluções possíveis para o problema, escolhendo a melhor delas como solução. O segundo deles é um algoritmo de Branch-and-Bound que também analisa extensivamente as soluções possíveis, mas se utiliza de uma função de limite superior das soluções sendo analisadas para evitar analisar soluções derivadas de uma solução que será inevitavelmente pior que a melhor solução conhecida até o momento.

Esse trabalho apresenta a descrição da implementação dos dois algoritmos, com seus detalhes de implementação e análise de complexidade, e também os resultados da execução desses algoritmos implementados para os casos de testes fornecidos, seguidos de uma análise.

2. Implementação

Foram implementados dois algoritmos que resolvem o problema da mochila binário de forma ótima: um algoritmo de backtracking e um algoritmo de Branch-and-Bound. Ambos os algoritmos foram implementados em Python 3, em um mesmo arquivo. Cada um deles foi implementado em uma classe e executado para as instâncias do problema fornecidas para teste. Também é gerado um arquivo CSV contendo o tempo de execução e solução encontrada pelos algoritmos em todas as instâncias de testes.

2.1. Algoritmo Backtracking

O algoritmo de Backtracking foi implementado de acordo com o pseudo-código apresentado em aula. O algoritmo foi escrito de forma recursiva e possui um código simples. Além disso, ele foi implementado com o cuidado de ser o mais eficiente possível, ou seja, são evitadas cópias e manipulações desnecessárias de listas e valores.

O seu funcionamento consiste em, começando do primeiro item da lista de itens, escolher colocar ou não o item na mochila e passar para o próximo item da lista. Quando uma solução é construída, ou seja, quando todos os itens são pegos ou não, é verificado

se as escolhas feitas geram uma solução válida e se a solução encontrada é a melhor até o momento e, caso seja, essa solução é salva como a resposta atual. Após a recursão terminar, ou seja, após todas as soluções possíveis terem sido geradas e analisadas, a melhor solução encontrada é a resposta ótima para o problema.

É fácil ter a intuição de que o algoritmo implementado é exponencial. Essa intuição pode ser confirmada ao percebemos que a recursão gera uma árvore binária completa, já que cada item analisado gera dois cenários diferentes: um em que ele é colocado na mochila e outro em que ele não é colocado. Como a árvore binária completa gerada pela recursão tem, trivialmente, uma altura N, sendo N o número de itens, ela possui 2^N nós, portanto a complexidade do algoritmo é da ordem de $\Theta(2^N)$. Vale notar que esse limite exponencial do tempo de execução é um limite tanto inferior quanto superior para o algoritmo, já que ele sempre gera toda a árvore de soluções.

2.2. Algoritmo Branch-and-Bound

O algoritmo Branch-and-Bound também foi implementado de acordo com o pseudo-código apresentado em aula. Ele também é simples, apesar de ser um pouco mais extenso que o algoritmo de Backtracking, sendo construído de forma recursiva. O algoritmo também foi implementado com o cuidado de ser o mais eficiente possível, evitando cópias e manipulações desnecessárias de listas e valores, além da ordenação dos itens para facilitar o cálculo da função de limite superior.

O seu funcionamento é similar ao funcionamento do Backtracking, com apenas uma diferença principal: a cada item analisado é computada uma função de limite superior da solução parcial gerada até o momento. Com essa função algumas soluções deixam de ser analisadas, pois elas sabidamente não possuem um custo melhor do que o custo da melhor solução encontrada quando a solução está sendo computada. Outra diferença com o algoritmo do Backtracking, que é consequência da primeira, é que antes da execução da recursão a lista de itens é ordenada de acordo com a função $\frac{v_i}{w_i}$, sendo v_i o valor do item e w_i o seu peso. Isso é feito para facilitar o cálculo da função de limite superior.

Portanto, o funcionamento do algoritmo consiste em, a partir do primeiro item da lista, computar a função de limite superior e verificar se o seu valor não é inferior à melhor solução encontrada até o momento, caso alguma solução já tenha sido computada. Caso o valor da função seja maior, a função é chamada recursivamente para o próximo item duas vezes, uma adicionando o item atual na solução e outra sem adicionar o item atual. Quando uma solução é construída, assim como no Backtracking, é verificado se a solução é válida e se ela é a melhor solução encontrada até o momento. Caso ela seja, ela é salva como a resposta atual. Após o fim da função recursiva, a melhor solução encontrada é a resposta ótima para o problema.

Também é fácil ter a intuição de que esse algoritmo apresenta um comportamento exponencial. No entanto, também é possível notar que ele parece ser mais eficiente do que o algoritmo de Backtracking, já que ele pode descartar um grande número de soluções, evitando assim diversas chamadas recursivas, diminuindo o tempo de execução. Podemos perceber que o algoritmo é exponencial no pior caso com a mesma analogia utilizada para o Backtracking. No Branch-and-Bound a árvore binária gerada pelas chamadas recursivas não é necessariamente completa, já que são realizadas podas, mas ela pode ser completa. Por isso, no pior caso, a complexidade do algoritmo é da ordem de $O(2^N)$.

Mas a árvore pode sofrer podas o suficiente para reduzir consideravelmente essa complexidade, por isso é um limite superior, mas não um limite superior e inferior, como no caso do Backtracking.

Por fim, a função de limite superior implementada consiste em ocupar o restante da mochila de forma fracionária com o melhor item ainda não visitado. Como os itens estão ordenados pela função $\frac{v_i}{w_i}$, o item sendo analisado no momento é o melhor item ainda não visitado. Assim, podemos definir o limite superior como $(W - w) * (\frac{v_i}{w_i}) + v$, sendo W a capacidade total da mochila, w o peso dos itens pegos na solução até agora, v o valor dos itens pegos até o momento, v_i o valor do item sendo analisado e w_i o peso do item sendo analisado. Dessa forma, simulamos uma mochila fracionária, que é preenchida com o melhor item disponível para ser pego, o que é trivialmente um limite superior para o problema da mochila binário considerando o espaço restante na mochila e os itens ainda disponíveis para ser pegos.

3. Resultados Testes

Primeiramente, vale notar que os resultados foram gerados em um computador com um processador i3-9100f e 16GB de RAM. Como citado anteriormente, ambos os algoritmos foram implementados em um mesmo arquivo, em Python 3, e foram executados utilizando o interpretador Python 3.9.6.

A execução de testes foi dada da seguinte forma: é aberto o arquivo CSV entregue com o trabalho, em que os resultados são salvos, e o seu cabeçalho é escrito nele, em seguida são listados os arquivos com as instâncias de testes e cada uma delas é aberta, lida. Assim que uma instância é lida, é executado o algoritmo de Backtracking, e em seguida, o Branch-and-Bound para ela. O tempo é medido para ambos os algoritmos quando são executados e, por fim, o resultado da execução deles é escrita no arquivo CSV. Após isso a próxima instância é executada, até que todas elas tenham sido resolvidas pelos dois algoritmos.

Os resultados para a execução dos testes são apresentados na tabela a seguir.

Instância	**Backtrack**		**Branch and Bound**	
	Tempo (ms)	**Solução**	**Tempo (ms)**	**Solução**
f1_l-d_kp_10_269	1.003	295.0	0.998	295.0
f2_l-d_kp_20_878	912.069	1024.0	16.029	1024.0
f3_l-d_kp_4_20	0.0	35.0	0.0	35.0
f4_l-d_kp_4_11	0.0	23.0	0.0	23.0
f5_l-d_kp_15_375	25.998	481.069	2.005	481.069
f6_l-d_kp_10_60	0.992	52.0	1.003	52.0
f7_l-d_kp_7_50	0.0	107.0	0.0	107.0
f8_l-d_kp_23_10000	6700.998	9767.0	7521.979	9767.0
f9_l-d_kp_5_80	0.0	130.0	0.0	130.0
f10_l-d_kp_20_879	865.610	1025.0	15.997	1025.0

Tabela 1. Resultados dos Testes

Podemos perceber, primeiramente que todos os algoritmos encontraram a solução ótima, como esperado. Somente na instância 5 que o algoritmo de Branch-and-Bound encontrou um resultado melhor, com uma diferença da ordem de 10^{-12}, o que pode ser atribuído à falta de precisão do ponto flutuante. Assim, podemos concluir que ambos os algoritmos encontram de fato a solução ótima.

Outro fato que podemos notar é o crescimento acentuado do tempo de execução para instâncias com mais itens, o que pode ser visto como uma confirmação empírica das complexidades exponenciais definidas e explicadas na descrição dos algoritmos. A diferença do limite superior e inferior de 2^N do Backtracking e do limite apenas superior de 2^N do Branch-and-Bound também pode ser observada empiricamente em alguns casos. Mas ao analisar essa diferença assintótica deve-se levar em conta que os dois algoritmos possuem constantes diferentes já que o Branch-and-Bound executa um cálculo a mais a cada recursão, o cálculo da função de limite superior.

Percebemos uma dominância do algoritmo de Branch-and-Bound no tempo de execução nas instâncias 1, 2, 5 e 10, sendo que nas instâncias 2, 5 e 10 a diferença de tempo foi considerável. Isso é explicado pelas soluções podadas do Branch-and-Bound que podem levar esse algoritmo a ter resultados consideravelmente melhores, em termos de tempo de execução.

Nas instâncias 6 e 8 percebemos um desempenho pior do Branch-and-Bound em relação ao Backtracking, sendo que na instância 8 essa diferença é mais relevante. O desempenho melhor do Backtracking nessas instâncias pode ser explicado pelo custo de computar a função de limite superior somado a poucas soluções podadas pelo Branch-and-Bound, o que o leva a seu pior caso, em que ele possui o mesmo custo do Backtracking, com o custo adicional de calcular a função de limite superior.

Já nas instâncias 3, 4, 7 e 9 o tempo de execução dos dois algoritmos foi zero pois esses casos possuem poucos itens, logo executam muito rapidamente, não sendo possível medir o tempo da execução com precisão.

Por fim, percebemos que a diferença de tempo entre os dois algoritmos é bem maior nos casos em que o Branch-and-Bound é melhor, ou seja, o Branch-and-Bound possui um potencial bem maior de encontrar a solução ótima em um tempo muito menor do que o Backtracking. Já quando o Backtracking se sai melhor, ele o faz com base em um custo constante, que é o custo de cálculo da função de limite superior, portanto ele não é capaz de executar muito mais rápido do que o Branch-and-Bound, já que no pior dos casos o Branch-and-Bound se degenera para um Backtracking com um custo constante adicional para cada chamada recursiva.

4. Conclusão

Primeiramente, podemos concluir com esse trabalho que o algoritmo de Branch-and-Bound é mais adequado para encontrar a solução ótima de uma instância do problema da mochila binário, em comparação ao algoritmo de Backtracking. Ambos os algoritmos são simples de se implementar e o Branch-and-Bound executa, na maioria das vezes, mais rápido que o Backtracking. Quando isso não acontece a diferença não é tão relevante quanto a diferença encontrado quando o Backtracking é mais lento.

Também podemos concluir que ambos os algoritmos são capazes de encontrar

a solução ótima, o que era esperado já que o Backtracking analisa todas as soluções possíveis e o Branch-and-Bound só descarta soluções sabidamente não ótimas.

Por fim, esse trabalho foi bom para entender a diferença prática, de tempo de execução e solução encontrada, entre os dois algoritmos, além de exercitar a comparação empírica de duas implementação de algoritmos diferentes.