UNIVERSIDADE FEDERAL DO PARANÁ

MARIA TERESA KRAVETZ ANDRIOLI (GRR20171602)

OTIMIZAÇÃO (CI 1238): TRABALHO 2

CURITIBA

2021

1 INTRODUÇÃO

Esse trabalho tem como objetivo fazer a leitura de uma entrada em formato de texto e transformar esses dados em um problema de otimização de algoritmos. A partir dessa transformação é possível utilizar estratégias já estudadas e documentadas para resolução do problema de forma a melhorar a performance dos algoritmos. Mais especificamente, esse trabalho trata do problema da mochila (knapsack problem [1]) usando a modelagem *branch and bound* [6].

2 PROBLEMA

2.1 DESCRIÇÃO

O problema específico que será tratado é o da Mochila Química, uma variação do problema da mochila [1], que é o seguinte: dados uma quantidade de itens, cada um com um peso e um valor, e uma mochila com uma capacidade máxima de peso, como preencher essa mochila de modo que o peso total não ultrapasse a capacidade máxima e o valor total seja o maior possível?

A particularidade da Mochila Química é que existem certos itens que não podem ser colocados juntos, então o problema se torna: dados uma quantidade de itens, cada um com um peso e um valor, uma mochila com uma capacidade máxima de peso e uma lista de itens que não podem ser colocados juntos na mochila, como preencher essa mochila de modo que o peso total não ultrapasse a capacidade máxima, o valor total seja o maior possível e nenhum dos itens proibidos estejam juntos?

2.2 MODELAGEM

O problema foi modelado usando o algoritmo *branch and boun*d [7]. O objetivo desse algoritmo é encontrar uma solução que consiga maximizar ou minimizar o valor de uma função entre um conjunto de soluções possíveis. O algoritmo é capaz de atingir tal objetivo recursivamente dividindo o espaço de busca (passo chamado de *branching*). Esse passo sozinho resultaria em um algoritmo de força bruta. Por causa disso, no *branch and bound*, é armazenado o *bound* (limite) de modo a eliminar soluções que não podem ser ótimas.

Tendo isso como base, no problema da mochila genérico a formulação do como um problema de otimização é a seguinte:

$$\text{maximize} \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \sum_{j=1}^{n} w_j x_j \leq W,$$

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

Imagem retirada de [8]

No qual existem *n* itens (*1* <= *j* <= *n*), cada um com um valor *p* e um peso *w*. Além disso, cada *x* pode ter valor de 0 ou 1, caso o item seja ou não selecionado, respectivamente. *W* é a capacidade máxima da mochila. A partir disso, é feita uma pequena modificação de modo a considerar os pares proibidos.

3 IMPLEMENTAÇÃO

Para a implementação, primeiro é feita a leitura de todos os dados de entrada, sendo eles o número de itens, o número de pares proibidos, a capacidade de mochila como inteiros na primeira linha. Na segunda linha, temos os pesos dos n itens. Na terceira linha, temos os valores dos n itens. Por fim, linhas por linha, temos em cada uma um par proibido.

Depois de serem feitas as leituras, é feita a modificação das estruturas para que as tenhamos do melhor modo a serem colocadas no algoritmo de resolução do problema. Nessa modificação, é criada uma lista (chamada V) onde cada elemento possui o valor, o peso e uma lista de índices de itens com os quais o item atual não pode interagir. É em cima dessa lista que o algoritmo de resolução do problema age.

Esse algoritmo faz o seguinte:

1. Ordena o V em ordem decrescente da divisão do valor pelo peso de cada item.
2. Inicializa lucro máximo com 0.
3. Cria uma fila FIFO (F) vazia e insere nela um item com todos os valores zerados. Esse item será usado para percorrer a árvore de decisão.

4. Percorre em loop a árvore enquanto a fila F não estiver vazia fazendo:
    a. Remove um item de F.
    b. Calcula o lucro do nodo do nível seguinte. Se for maior que o lucro máximo, atualiza o máximo.
    c. Calcula o limite (bound) do nodo do nível seguinte. Se for maior que o lucro máximo, adiciona o nodo do nível seguinte em F.
    d. Lida com o caso em que o próximo nodo não faz parte da solução e o adiciona em F.

Esse algoritmo utiliza uma estratégia gulosa, já que calcula a divisão do valor pelo peso e usa essa informação para tomar decisões. É isso que garante que a solução encontrada seja a ótima.

4 EXEMPLO

4.1 PRIMEIRO EXEMPLO

No exemplo, há uma solução e o algoritmo é capaz de encontrá-la:

```
Entrada:
8 4 10
2 1 3 5 6 2 9 4
1 1 2 6 10 3 100 90
1 2
2 4
5 8
6 3
Saída padrão:
101
2 7
Saída de erro:
Tempo: 0.00013 segundos
Nós: 2
```

Ou seja, o máximo lucro é 101, com os itens 2 e 7 sendo escolhidos.

4.2 SEGUNDO EXEMPLO

No exemplo, não há uma solução e o algoritmo então retorna 0 como lucro máximo e nenhum índice de item:

```
Entrada:
7 2 2
4 3 6 4 3 10 9
10 20 43 2 4 1 90
3 4
1 2
5 6
Saída padrão:
0
Saída de erro:
Tempo: 0.00008 segundos
Nós: 0
```

Ou seja, o máximo lucro é 0.

5 EXECUÇÃO

make gera o executável *quimica*

./quimica < **arquivo_de_teste**

6 GITHUB

Link do repositório no github contendo todos os arquivos do projeto:
https://github.com/mariaandrioli/otimizacao-t2

REFERÊNCIAS

[1] “Knapsack Problem.” *Wikipedia*, Wikimedia Foundation, 27 Nov. 2021, https://en.wikipedia.org/wiki/Knapsack_problem.

[2] Saurabhschool, director. *YouTube*, YouTube, 2 Nov. 2013, https://www.youtube.com/watch?v=slayHO7gKEQ. Accessed 5 Dec. 2021.

[3] Trivedi, Utkarsh. “Branch and Bound Algorithm.” *GeeksforGeeks*, 15 Nov. 2021, https://www.geeksforgeeks.org/branch-and-bound-algorithm/.

[4] Bettinelli, Andrea & Cacchiani, Valentina & Malaguti, Enrico. (2017). A Branch-and-Bound Algorithm for the Knapsack Problem with Conflict Graph. INFORMS Journal on Computing. 29. 457-473. 10.1287/ijoc.2016.0742.

[5] Trivedi, Utkarsh. “Branch and Bound Algorithm.” *GeeksforGeeks*, 15 Nov. 2021, https://www.geeksforgeeks.org/branch-and-bound-algorithm/.

[6] Trivedi, Utkarsh. “Implementation of 0/1 Knapsack Using Branch and Bound.” *GeeksforGeeks*, 26 Nov. 2019, https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/.

[7] “Branch and Bound.” *Wikipedia*, Wikimedia Foundation, 7 July 2021, https://en.wikipedia.org/wiki/Branch_and_bound.

[8] “List of knapsack problems.” *Wikipedia*, Wikimedia Foundation, 7 November 2021, https://en.wikipedia.org/wiki/List_of_knapsack_problems.