Implementação do Problema da mochila química

Ovidio Jose da Silva Junior

28 de novembro de 2021

Resumo

Este trabalho analisa e implementa uma solução para o problema da mochila química, derivado do problema da mochila clássico.

1 Introdução

1.1 Branch and Bound

O problema branch and bound é um método enumerativo para otimização de problemas de custo e disponibilidade, seu objetivo é o particionamento de um espaço de soluções utilizando de técnicas limitantes que auxiliam na ramificação do problema, ou seja, com o branch and bound ao invés de explorar todo o conjunto viável de possibilidades para a resolução do problema, particiona-se o problema original, as resultantes podem ser categorizadas e resolvidas, até que se chegue em um subproblema com maior valor máximo, e que corresponda as restrições do problema original(Oliveira,2003).
O branch and bound pode ser interpretado como uma arvore binaria, da qual a raiz representa todo o conjunto possível do problema inicial, as ramificações vão surgindo a medida que o problema pode ser particionado, esse particionamento ocorre com o **relaxamento** do problema inicial, sendo uma solução que corresponde ao limite superior e outra ao limite inferior da função reparticionada, assim segue consecutivamente até que não se haja mais possibilidades de ramificação, a solução ótima será a do folha que obtiver o maior valor respeitando as restrições dadas pelo problema original, ou seja uma solução viável da raiz.

1.2 Problema da mochila

O problema da mochila pode auxiliar numa visão maior do que seria o método branch and bound.
Vamos supor que existe uma mochila com um determinado limite de peso, e existe diversos itens que precisamos colocar nessa mochila à maximizar o valor dos itens que vamos levar para por exemplo vender esses itens.Para esse problema inicial temos duas coisas das quais se importar. O limite pré estabelecido da mochila assim como colocar nessa mochila uma combinação de itens que garanta o maior lucro possível.

1.3 Modelagem

Podemos iniciar supondo valores pesos para esses itens que serão nomeados de x1 até Xn, para iniciar teremos a função máxima como:
$Max = x1.p1 + x2.p2 + x3.p3 + x4.p4 + x5.p5$
Supondo valores arbitrários para o peso e os valores dos itens teremos que:
$v1 = 2, 2 = 5, v3 = 9, v4 = 5, v5 = 50$
e os pesos que eles ocupam como:
$p1 = 1, p2 = 2, p3 = 5, p4 = 10, p5 = 20$
se nossa bolsa tiver um tamanho igual a 15, e enumerarmos todas as possibilidades teremos como soluções, sendo cada elemento representante de um item: $\{0, 0, 0, 0, 0\}$, $\{0, 0, 0, 0, 1\}$, $\{0, 0, 0, 1, 0\}$, $\{0, 0, 1, 0, 0\}$
$\{0, 1, 0, 0, 0\}$, $\{1, 0, 0, 0, 0\}$, $\{0, 0, 0, 1, 1\}$
, $\{0, 0, 1, 1, 1\}$, $\{0, 1, 1, 1, 1\}$ ... $\{1, 1, 1, 1, 1\}$
Serão 2^5 formas de resolver o problema, porem, como pose ser visto acima, existem possibilidades que

não serão viáveis como por exemplo $\{0,0,0,0,1\}$, pois o item $x5$ passa do tamanho total da bolsa, e existem também os casos como o $\{0,0,0,0,0\}$, que seria possível, mas não teria vantagem nenhuma não levar nenhum item, então a partir daqui poderemos ter uma visão do branch and bound e começar a cortar essas tentativas que não serão benéficas para o problema principal, que é carregar o maior numero de itens que maximize o lucro.

1.4 Função Limitante

As limitações do problema serão, não estourar a bolsa, e maximizar o lucro, então, é inteligente começar as tentativas pegando os itens com melhor relação entre valor e peso, inicia-se fazendo essa relação que definira a ordem que esses itens entrarão: $r1 = 2, r2 = 2,5, r3 = 1,8, r4 = 0,5, r5 = 27,5$
Na primeira possibilidade, $x5$ tem a maior relação, porem ele não cabe na mochila, o próximo item testado será o $x2$, que satisfaz a restrição do problema, agora resta na machila 10 para se ocupar na mochila pois o item 2 tem peso 5.
Em sequencia vem o item $x1$, que ocupará 2 espaços na mochila restando agora 8 espaços para ocupar. O próximo será o $x3$, poŕem neste caso ele não cabe inteiramente na função da maneira como ela já está organizada, mas se a mochila estivesse vazia ele poderia ser uma opção, então aqui entra a técnica de relaxamento da função, onde será particionado o problema em 2, onde o $x3$ é obrigatoriamente 1, e onde ele é obrigatoriamente zero, os pisos superiores e inferiores. Essas duas partes serão enxergadas como folhas do problema inicial, onde será refeito o preenchimento da bolsa seguindo as restrições do problema original. Antes de ramificar, é importante entender que a arvore não pode crescer infinitamente, e o que torna o algoritmo branch and bounch diferente dos outros métodos de enumeração é exatamente esse controle na distribuição do problema. Ele trabalha com cortes, a seguir é explicado esses métodos.

- **inviabilidade**, quando ocorre uma particionalização do problema, pode ocorrer de uma das vertentes apresentarem características que não são permitidas pelas restrições originais e tão pouco pelas restrições de relaxamento, quando cai neste caso o ramificação no galho é cortado por inviabilidade, pois a partir daquele galho os novos nós iriam fazer parte de um novo subconjunto restringido com as características inviáveis.

- **integralidade**, quando a ramificação chega em uma folha com solução viável para o problema original.

- **limitação**, quando um ramo tem um valor máximo que é menor a uma solução já obtida em outro ramo, já que abaixo desta folha os valores tendem a diminuir com a limitação dos espaços amostrais disponíveis para combinação e solução do problema.

Seguindo para a ramificação temos:

- $x3$ como obrigatoriamente 1
  - $x3$ Como $x3$ já está alocado na bolsa, o valor disponível na mochila será 6, e prosseguindo para a escolha do item com maior valor, vem o $x2$, restando agora na mochila apenas 1 espaço.
  - O próximo item seria o $x1$, porem ele não cabe no cenário atual, então será particionado novamente o problema. Como esse particionamento já se encontra em um ramo, os valores fixados da folha acima permanecem, pois o espaço amostral é um subconjunto das folhas acima que representam outro subconjunto do problema original.
  - $x3$ fixado em 0 e $x1$ fixado em 0
    - Tendo 2 fixados em 0, o seguinte item da lista sera o $x2$, restando agora 10 espaços na bolsa
    - Essa ramificação agora se encerra por integralidade com seu ultimo elemento $x4$.
    - Calculando o máximo do conjunto final deste ramos temos um lucro máximo de 13.
  - $x3$ fixado em 0 e $x1$ fixado em 1
    - Como $x1$ fixado em 1, a mochila inicia com 13 espaços disponíveis, podendo pegar o próximo da lista, o $x2$, restando 8 espaços disponíveis.

    * O próximo item $x4$ finaliza a ramificação por integralidade. pois os itens escolhidos cabem integralmente na mochila.

- $x3$ como obrigatoriamente 0
  - Como o item três está zerado, a mochila pode ser preenchida normalmente seguindo a ordem dos item estabelecida anteriormente, sendo assim o item escolhido dessa vez será o $x2$, restando agora na mochila 10 espaços.
  - O próximo item da lista, levando em consideração a restrição de $x3$ é o item $x1$ que ocupa na na mochila 2 espaços, restando agora 8 espaços para serem preenchidos.
  - O próximo e ultimo item a ser preenchido, será o $x4$ que ocupa apenas 5 espaços na bolsa, restando apenas 3 espaços a serem ocupados, porem, não existe nenhum candidato que satisfaça as restrições, pois $x3$ está obrigatoriamente zerado e x5 né enviável sendo 1 pois é maior que o valor da mochila, sendo assim encerramos essa ramificação por integralidade, já que satisfaz todas as restrições do problema original.
  - a função máxima obtida com o vetor $\{1, 1, 0, 1, 0\}$ é 13.

Para este exemplo o máximo seria 13, que foi o maior valor alcançado na ramificação que respeita as restrições da raiz da arvore.

2 Problema da mochila química

E se além da restrição do tamanho da mochila, existisse a restrição de itens que não podem estar na mesma mochila por questões químicas, este trabalho se propõem a resolver este problema usando uma modelagem derivada do problema da mochila original.

2.1 Modelagem

Para este problema temos o seguinte exemplo a ser modelado: A bolsa tem capacidade 10, 5 itens que podem ser escolhidos e 3 pares de itens proibidos que não podem estar juntos na mesma mochila. Sendo o conjunto de pesos seguindo a ordem de elementos de 1 a 5 $\{10, 4, 3, 2, 11\}$ e os valores $\{10, 2, 2, 1, 6\}$ e os pares proibidos $\{(12), (13), (14)\}$, seguindo o mesmo fluxo da sessão 1.4, calculasse a relação peso e valor, resultando em $\{1, 0.5, 0.66, 0.5, 6\}$.

- O primeiro item da mochila será o item 5, restando 9 espaços na mochila.
- seguindo a ordem temos o item 1, como ele é maior do que o espaço disponível, o problema se particionará em 2, assim como mostrado na sessão 1.4.
  - Item 1 fixado em 0
    * Como o item 1 está fixado em 0, podemos partir para o próximo elemento, sem precisar ajustar o tamanho da mochila, sendo assim, o próximo item escolhido será o item 5, restando agora 9 espaços na mochila
    * o próximo item da lista será o item 3, restando 6 espaços para preencher
    * seguindo o fluxo temos o item 2 ou 4, as duas cabem na mochila, e podamos este ramo por integralidade, com função máxima 11.
  - item 1 fixado em 1
    * como item 1 está fixado regulamos o tamanho da mochila que estará cheia impossibilitando novas ramificações, este galho está podado por integralidade porquê satisfaz as restrições originais ainda que enchendo a bolsa sozinho, e tendo como função máxima 10.

A função máxima para esse exemplo é 11 pertencendo ao subconjunto $\{0, 1, 1, 1, 1\}$.

3 Implementação

Como o método Branch and Bound segue um algoritmo de resolução, conseguimos representa lo em um código. Neste trabalho será abordado a linguagem em python para a resolução do problema.

3.1 funções

3.1.1 Ramificação e corte da arvore

Para controlar a ramificação da arvore, assim como verificar os pontos citados na sessão 1.4, foi criado a função:

```
def branch_create(lista,maior):
```

Listing 1: Função de ramificação e corte

A função é recursiva, e recebe como entrada uma lista com os itens que identificarão futuramente os fixados em 0 e 1, também é a função responsável por chamar as outras funções que serão apresentadas a seguir.

3.1.2 Ordem de escolha

Como visto na sessão 1.4, o inicio da modelagem é a escolha do item a ser incluído na mochila, na função principal se calcula a relação peso e valor, e se ordena do maior para o menor.:

```
def acha_o_maior(j):
    global ordenado1
    if j>= len(ordenado1):
        return -1
    maior = ordenado1[j][1]
    return maior
```

Listing 2: Ordem de escolha

3.1.3 Restrição de par proibido

Como visto na descrição do problema no capitulo 2, a mochila química possui pares de itens que não podem compartilhar da mesma mochila, para garantir que isso não ocorra a função abaixo percorre a lista com os pares proibidos e faz a verificação, retornando um booleano que identifica a existência de um par na bolsa na bolsa ou não:

```
def par_proibido(item,mochila):
    achou = False
    global duplas
    for i in range(0,len(duplas)):
        if duplas[i] ==item:
            if(i%2==0):
                for j in range(0,len(mochila)):
                    if mochila[j]== duplas[i+1]:
                        achou =True
            else:
                for j in range(0,len(mochila)):
                    if mochila[j]== duplas[i-1]:
                        achou =True
    return achou
```

Listing 3: Restrição de par proibido

A função percorre verificando se algum item na mochila é o par proibido do item que será incluído na bolsa, ou vice-versa com os itens já inclusos na mochila

3.1.4 Controle de Restrições

A função responsável por verificar todas as restrições estabelecidas, como controle do tamanho da mochila, item já incluído na mochila e se existe um par proibido é a função abaixo.

```
def itensmochila(item,mochila,lista):
    global c
    global lista_escolhidos
    global valores_itens
    n = 0
    if item == -1:
        return False
    for i in range(0,len(mochila)):
        if item == mochila[i]:
            return True
        if lista[item-1] == 3:
            return True
        if valores_itens[item-1]>c:
            return True
        if (par_proibido(item,mochila)):
            return True
    return False
```

Listing 4: Controle Restrições

A função é chamada toda tentativa de inclusão de um novo item, caso o item não corresponda com as exigências, a função de ramificação fica responsável por pegar o próximo item da lista de possibilidade de itens.

3.1.5 Controle da mochila

A cada ramificação a mochila precisa se adaptar aos itens que estão fixados em 0 e 1, para isso a função abaixo é chamado em todo o inicio de ramificação.

```
def create_mochila(lista):
    mochila = list()
    global valores_itens
    global c
    j=0
    for i in range(0,len(lista)):
        if lista[i] == 2:
            if(itensmochila(i+1,mochila,lista)):
                return -1
            mochila.append(i+1)
            j = j+ valores_itens[i]
    if j > c:
        return-1

    return mochila
```

Listing 5: Controle da mochila

3.1.6 Controle do tamanho da mochila

Assim como a função acima faz para controlar os valores em cada ramificação, é necessário também arrumar a variável responsável pelo tamanho da mochila, a função a seguir resolve contando os itens dentro da mochila.

```
def tamanho_mochila(mochila):
    global valores_itens
    n = 0
    for i in range(0,len(mochila)):
        n = n+valores_itens[mochila[i]-1]
    return n
```

Listing 6: Controle do tamanho da mochila

3.1.7 Função de itens escolhidos

Responsável por preparar uma lista com os itens escolhidos para a função maxima

```
def create_results(mochila):
    global n
    result = [0]*n
    for i in range(0,len(mochila)):
        result[mochila[i]-1]=1

    return result
```

Listing 7: função de maximização

3.1.8 Função Máxima

Responsável por calcular o valor da função máxima, com base na lista final criada na função acima no final da ramificação.

```
def maximizador(lista):
    global pesos
    soma=0
    for i in range (0,len(lista)):
        soma = soma + pesos[i]*lista[i]
    return soma
```

Listing 8: função de maximização

4 Testes

Para executar os testes, foi adotado a metodologia de testes unitário, a separação das tarefas em funções facilitaram o uso reservado das chamadas e o controle do tempo e da ramificação pôde ser analisado com cautela, como nas sessões a seguir.

4.1 Testes Unitários

Cada função recebeu um teste isolado de funcionamento com um caso ideal, e utilizando da saia padrão foi possível analisar as saídas e encontrar os erros que estavam acontecendo durante a execução.A metodologia foi essencial para a criação da recursividade, no inicio era muito comum estourar a pilha com a função, era necessário ajuste função por função, e para que fosse possível essa analise foi limitado a ramificação em 10 nós para objetivos de testes.

4.2 Analise dos nós

Para contagem de nós foi utilizado uma variável global que servia como contador, a cada chamada da função de branch, ela somava 1, foi escolhido este método pois facilitaria na contagem de nos totais da arvore.
Em testes utilizando como o exemplo da sessão 2.1 se obteve o mesmo numero de nós, mas foram utilizados outros testes afim de analisar varias possibilidades.

- itens sem pares proibidos
- itens com mais de um par de proibidos
- entradas vazias
- nó isolado
- solução inviável.

A partir destes testes foi possível montar uma tabela, que consta o tempo de execução, o numero de nós, numero de nós inviáveis, numero de nós íntegros e nos podados por limitação:

Comparação de exemplos							
itens	pares	Capacidade	nós	íntegros	limitados	inviáveis	tempo de execução
3	1	5	1	1	0	0	1.69277191162e-05
8	4	10	11	6	1	4	0.000102996826172
4	0	10	9	5	3	1	8.9168548584e-05
5	3	10	3	3	0	0	4.79221343994e-05
3	1	1	1	1	0	0	1.50203704834e-05
15	15	10	151	82	40	29	0.00493192672729
3	1	1	1	1	0	0	1.31130218506e-05

Como pode ser visto, os exemplo com maior numero de nós correspondem ao maior tempo de execução, este tempo aumenta dado ao maior espaço de resultados possíveis que são particionados em subproblemas, isso pode variar com as limitações, como os pares proibidos, ou o tamanho dos itens.

4.3 Execução

Para execução do programa e a geração do executavel quimica, foi utilizado o modulo Pyinstaller, instalado com o comando:

```
pip install pyinstaller
```

Listing 9: Instalação do pyinstaller

após a instalação foi gerado o executável com o comando:

```
python2.7 -m PyInstaller --onefile  quimica.py
```

Listing 10: geração executável

Assim, há possibilidade de funcionamento do programa em outras plataformas, utilizando arquivos como entrada.

5 Conclusão

O algoritmo Branch and Bound, apesar de ainda pertencer à um grupo de métodos de tentativa e erro, ainda é uma ótima opção, comparada aos outros métodos de enumeração, a analise contribuiu muito para o aprendizado, além de cativar o estudo das ferramentas de resolução de otimização de problemas e soluções.

Referências

OLIVEIRA, Devair; Cardoso,Raul Henriques. ALGORITMOS BRANCH AND BOUND PARALELOS: RESULTADOS DE EXPERIMENTOS COMPUTACIONAIS. Universidade Federal do Espírito Santo. XXCV SBPO, Natal-RN, 2003.