Relatório referente a etapa 2

Iago Floriano[1], **Tiago Serique**[1]

[1]Universidade Federal do Paraná, Curitiba, Brazil

{imf19, tsv19}@inf.ufpr.br

1. Recursos usados

Para a realização do trabalho foram utilizados a linguagem de programação Python3 devido sua facilidade para manipulação de dados, e a biblioteca Gurobi Optimizer como resolvedor de problemas lineares relaxados. Afim de facilitar a manipulação e execução, foi criado um makefile. Para instalar as dependências necessárias basta usar o comando ”make install”, e para gerar o executável do código use ”make”.

2. Definindo o problema:

O problema apresentado é similar ao problema de empacotamento de caixotes (bin packing), porém usado em um contexto diferente. Neste problema, o contexto é o de um caminhão que precisa levar alguns itens ao seu destino, e deseja-se minimizar o número de viagens que este caminhão faz.

O caminhão possui uma capacidade de peso C. Existem n produtos, de I_1 até I_n, e cada um desses produtos tem um peso P_i associado a eles.

Neste problema é necessário que cada item esteja em uma viagem, com cada viagem sendo numerada de 1 a n, e que nenhuma das viagens do caminhão tenha itens com pesos somados maiores que a capacidade. Além disso, existem pares ordenados de números, ex. (3, 2), que indicam a ordem em que os itens devem ser entregues. Nesse caso, seria necessário que o item 3 seja mandado em uma viagem antes do item 2.

3. Modelagem do problema

O problema será resolvido de forma a usar programação linear, portanto é importante fazer uma modelagem do problema.

minimizar $\sum_{i=1}^{n} v[i]$

Sujeito a:

$\sum_{j=1}^{n} IV[i][j] = 1 \; \forall i \,/\, 1 \leq i \leq n$, cada item deve ser colocado em apenas uma viagem.

$\sum_{i=1}^{n} IV[i][j] * w[i] \leq C * v[j] \; \forall j \,/\, 1 \leq j \leq n$, o peso de cada viagem não pode ser maior que a carga do caminhão.

$v[i] \in \{0,1\} \; \forall i \; 1 \leq i \leq n$ ou uma viagem acontece ou não acontece.

$IV[i][j] \in \{0,1\} \; \forall i,j/1 \leq i,j \leq n$ item i está na viagem j ou não está.

$\sum_{j=1}^{n} IV[i_1][j] > IV[i_2][j] \; / \; \forall(i_1,i_2) \in P$ restrição de ordem dos pares ordenados

Variáveis usadas:
v, um vetor para saber se a viagem i aconteceu. Sendo $v[i] = 1$ caso viagem aconteceu, e $v[i] = 0$ caso contrário.

C, a carga máxima do caminhão

w, um vetor com os pesos dos itens, sendo $w[i]$ o peso do item I_i

n, o número de itens a serem levados, também é o número máximo de viagens que pode ocorrer (cada item precisando de uma viagem com apenas aquele item)

IV, uma matriz em que $IV[i][j]$ representa se o item I_i está na viagem $v[j]$

P, o conjunto de pares ordenados da restrição de ordem das viagens

4. Resolução do problema

Para a resolução do problema será usado um algorítimo de branch & bound. Cada nó da árvore gerada pelo algoritmo de branch and bound feito será representado como um vetor v, esse vetor será calculado de forma que $v[i]$ será igual a viagem onde o item i será colocado (o vetor será inicializado com todas as posições sendo 0)

As chamadas recursivas são feitas a tentar colocar o máximo de itens nas viagens em ordem crescente (primeiro se coloca o máximo de itens na primeira viagem, depois o máximo de itens na segunda, etc). A função do branch and bound é dada da seguinte forma.

$$b\&b = \begin{cases} max(v) & \text{, se } 0 \notin v. \\ min\Big(min\Big(b\&b(v', atual)\Big), min\Big(b\&b(v'', atual+1)\Big)\Big) & \text{, caso contrário} \end{cases}$$

Em min$\Big($b&b(v', atual)$\Big)$, onde v' = v e v'[i] = atual $\forall I_i$ que pode ser adicionado a viagem atual. Em min$\Big($b&b(v'', atual + 1)$\Big)$, onde v'' = v e v''[i] = atual + 1 $\forall I_i$ que não pode ser adicionado a viagem atual, mas pode ser adicionado a viagem atual + 1.

Sendo assim para se obter a uma resposta ótima basta usar o resultado de $b\&b(v, 1)$ onde $v[i] = 0 \; / \; \forall i \in \{1..n\}$ onde n é o número de itens. O número de viagens necessárias será $max(v)$.

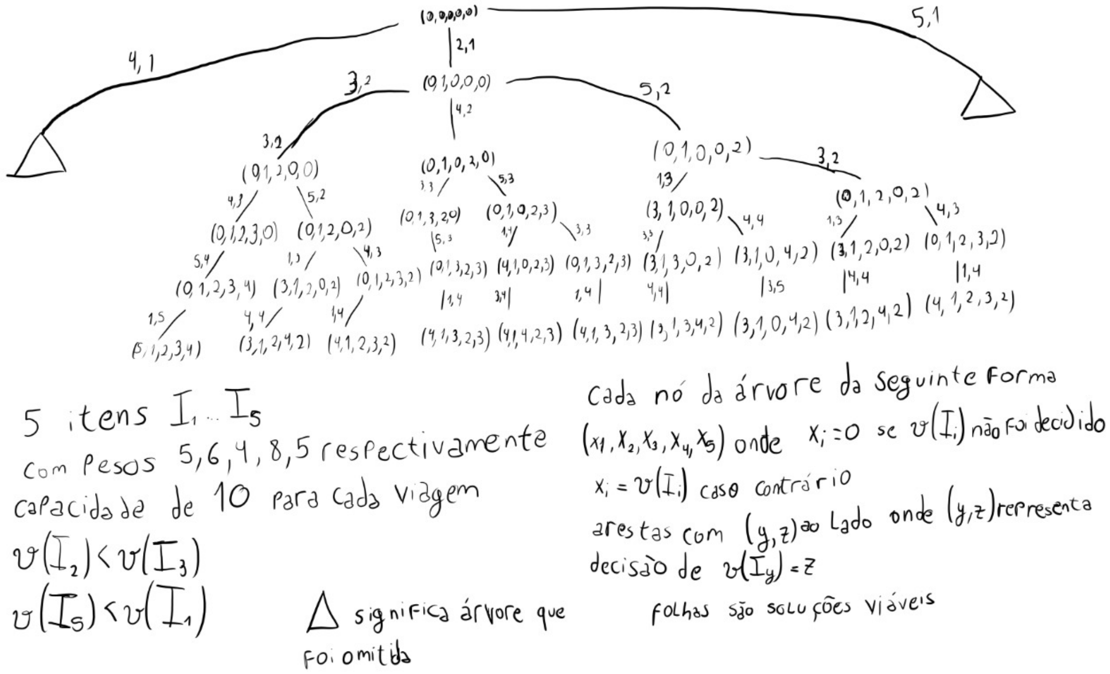

Figura 1. Exemplo de árvore de execução

Para que o branch and bound não seja simplesmente um backtracking se faz o uso de uma função otimista que da um resultado aproximado para soluções parciais do problema. Seja p o resultado da função otimista da solução parcial de um nó. Se p é maior que o melhor valor obtido durante a exploração da árvore, não será explorado nenhum filho desse nó. Esse valor de p é calculado para todo nó explorado na árvore.

5. Primeira tentativa de função otimista

A estimativa desse problema foi feita tirando algumas restrições que causam com que o problema tenha resolvimento muito demorado e relaxando outras restrições para que admitissem valores reais.

As restrições removidas foram as seguintes:
$v[i] \in \{0, 1\}$ virou a restrição $v[i] \in R \ / \ 0 \leq v[i] \leq 1 \ \forall i \ 0 \leq i \leq n$

$IV[i][j] \in \{0, 1\}$ virou a restrição $IV[i][j] \in \ R \ / \ 0 \leq V[i][j] \leq 1 \forall i, j \ / \ 1 \leq i, j \leq n$

a restrição $\sum_{j=1}^{n} IV[i_1][j] > IV[i_2][j] \ / \ \forall (i_1, i_2) \in P$ foi removida, pois seriam muitas restrições utilizadas na utilização da biblioteca Gurobi, fazendo o cálculo dessa estimativa muito lento.

Além dessas restrições que foram alteradas/removidas, foi adicionado restrições para aceitar soluções parciais

$IV[i][j] = 1, \forall (i, j) \in parcial$, sendo $parcial$ um conjunto de pares correspondentes à soluções parciais (i, j) significa que i está na viagem j.

Após testes, foi notado que o resultado desse relaxamento não é útil, pois sempre retorna $\frac{\sum_{i=1}^{n} w[i]}{C}$. Portando no código está simplesmente sendo feito o calculo desse valor sem usar programção linear

6. Exemplo usado:

Como entrada, foi utilizado o exemplo disponibilizado, onde a quantidade de itens é 5, com pesos 5, 6, 4, 8, 5, a capacidade do caminhão de 10, os pares ordenados (2, 3) e (5, 1), e uma solução parcial com 2 pares, onde a viagem do item 3 seria a de número 2 e a viagem do item 5 seria a de número 3. O resultado desse exemplo é igual a 2,8 viagens, o mesmo valor da saída do programa em feito com o resolvedor gurobi.

7. Referências bibliográficas:

Understanding and Using Linear Programming. Jiří Matoušek, Bernd Gärtner. 2007.

Gurobi Optimization. Disponível em: `https://www.gurobi.com/documentation/9.5/quickstart_mac/cs_grbpy_the_gurobi_python.html`. Último acesso em: 13/04/2022.