Relatório do Primeiro Trabalho
Disciplina de Otimização

Marllon Weslley Cabral Marques
mwcm17@inf.ufpr.br
GRR20170149

Junho de 2021

1 Introdução ao Problema

Uma empresa aluga máquinas (para uso remoto) sob demanda de seus clientes. A única restrição é que as máquinas só podem ser usadas durante um mesmo dia de trabalho (expediente tem duração de 9 horas, o mesmo que 540 minutos). Existe a possibilidade de mais de um destes usos poderem ser alugados em um mesmo dia para uma mesma máquina, se a soma dos tempos for menor que as 9 horas do expediente. Cada cliente pede quanto tempo, em minutos, vai usar uma máquina. Esse tempo deve estar entre 0 e 540 minutos.

A empresa tem m máquinas. Ao receber um conjunto de pedidos, o gerente da empresa precisa escalonar em qual máquina e em qual dia cada uso vai ser feito. Considere que a demanda (pedidos) é dada por um conjunto de n pares de formato (n_i,t_i), onde n_i é o número de pedidos de tempo t_i, como (1 $\leq$ i $\leq$ n). Queremos minimizar o número de dias necessário para atender aos pedidos da demanda.

2 Modelagem

Do enunciado do problema, podemos considerar que cada máquina pode executar uma combinação de tempos t_i para resolver os pedidos. Essa característica se assemelha ao problema da mochila (*Knapsack problem*), no qual você possui uma lista de itens que deseja armazenar em uma mochila, a combinação destes itens não pode ultrapassar a capacidade de peso da mochila.

Definição 2.1 (m) *Conjunto de máquinas disponíveis na empresa.*

Definição 2.2 (n) *Quantidade total de pedidos solicitados.*

Definição 2.3 (T) *É o conjunto de todos os tempos fornecido na entrada.* $\forall t_i \in T$*,* $t_i \leq 540$*.*

Definição 2.4 (n_i, t_i) *Tupla do conjunto de dados quantidade de pedidos* (n_i) *para um determinado tempo* (t_i)*. Onde* ($1 \leq i \leq n$) *e* ($0 < t_i \leq 540$)*.*

Definição 2.5 (P) *Conjunto que possui a combinação com repetição do conjunto de pedidos de tempo, onde:*

$$P = \binom{t}{i}$$

Definição 2.6 (p) *São os padrões compostos das combinações dos tempos de pedido, onde* $p \in P$*, onde a soma de todos os elementos de tempo presentes na combinação p, tenham o formato de* $1 \leq p \leq 540$*.*

Definição 2.7 (x_i) x_i *representam a quantidade de vezes que um tempo* t_i *pode aparecer para o conjunto de quantidade de pedidos daquela instancia do problema.*

Uma das faces do problema consiste em encontrar combinações de tempo de forma a minimizar a quantidade de tempo que vai ser desperdiçado nas máquinas por dia de trabalho. Essas combinações de tempo devem gerar equações que irão ser utilizadas pelo programa solucionador lp_solve.

2.1 Restrições

Após a compreensão do problema conseguimos saber quais são as incógnitas presentes no problema, e portanto agora podemos começar a modelagem das equações de restrição que irão colaborar com a geração da saída do programa que tera o formato de entradas para o programa linear que irá resolver o problema *lp_solver*. Nesta proposta de modelagem foi utilizado as regras descritas a baixo que serviram como base para modelagem das equações de restrição:

1. A soma dos tempos disponíveis em cada uma das combinações de tempos de utilização não pode ser maior que 540 para nenhuma das máquinas;
2. A soma das combinações de tempo são comutativas;
3. As combinações que tiverem tempo sobrando maior ou igual ao menor tempo de pedido fornecido, não vão estar presentes no conjunto de equações;

A primeira regra descrita acima serviu como base de parada para o gerador de combinações com repetição utilizado no código, no momento em que a primeira combinação tivesse a soma de todos os elementos maior que 540, o gerador deveria parar de fornecer combinações.

$$\forall p \in P \in \binom{n}{i} \leq 540 \tag{1}$$

Após observado que a posição dos diferentes tipos de tempo não importavam na combinação, a segunda regra surgiu para garantir que não ocorra repetições das equações de restrição.

A terceira regra esta presente para garantir que cada computador seja utilizado ao seu máximo diariamente, de forma com que independente da combinação de tempos mostrada na Equação 2, o valor que sobrar de um dia de trabalho vai ser menor do que o menor valor de tempos de pedido disponíveis. Dessa Forma, podemos garantir que as equações de restrição estão buscando sempre as combinações que otimizam o uso do tempo de dia de trabalho.

$$p = 540 - \sum C_i * t_i > min(T) \tag{2}$$

Onde C_i representa a quantidade de vezes que o tempo t_i aparece naquela combinação e $C_i \in \mathbb{Z}$, $C_i \geq 1$.

2.2 Função objetivo

A função objetivo que desejamos obter é construída em cima das informações das combinações $p \in P$, que descrevem os pedidos a serem rodados em forma de trabalho dia-máquina. Essas combinações p são representadas como x_i. Portanto a função objetivo que desejamos é a representação do mínimo de dias-máquina necessários para concluir o pedido de entrada. Logo, ela pode ser representada como:

$$min \sum_{i=1}^{|p \in P|} x_i \tag{3}$$

3 Implementação

O programa desenvolvido para este trabalho foi implementado em *Python 3*, fazendo uso da biblioteca *itertools*. Essa biblioteca fica responsável pela parte de geração das combinações com repetição dos padrões de tempo. O código fonte do trabalho pode ser encontrado no arquivo *main.py*.

3.1 Leitura de dados

A leitura das entradas para execução do programa é realizada utilização a entrada padrão (*stdin*), os dados fornecidos vão ser então lidos pela função tratamento, a qual fica repensável por ler linha por linha do arquivo e segmentar os valores de forma com que cada valor de entrada seja atribuído para um índice de uma lista. Essa lista vai passar por uma função de verificação de integridade, e durante a verificação vai ser checado se os valores de tempo inseridos estão dentro da margem permitida ($t_i \leq 540$), após a verificação é montado um dicionário com as chaves sendo os mesmos valores de tempo t_i e o valor da chave é dado por uma tupla com formato (n_i,t_i). Caso os valores estejam dentro das especificações permitidas para execução então vai ser dado continuidade a execução do problema, caso contrário é impresso mensagens de erro na saída padrão (*stdout*).

3.2 Criação das combinações

Dado os valores de entrada tratados durante a execução do problema, é na função *combinacao_valores* que vai ocorrer a computação das combinações e as fases de tratamento que vão gerar os padrões. O caminho de execução dessa função vai seguir a seguinte ordem:

1. Gera todas as combinações de tamanho $1 \leq i \leq n$, de forma que só pare quando a primeira soma de combinações for maior que 540;

2. Retira as combinações que são iguais. Exemplo: [200, 200, 100] = [100, 200, 200] = [200,100,200];

3. Realiza a montagem de novas combinações que ainda disponham de espaço de tempo para encaixe de um novo valor;

4. Realiza a simplificação das combinações, fazendo a contagem de quantas vezes cada elemento aparece na sequencia. Exemplo: [200, 200, 100] = [2, 200, 1, 100];

5. Realiza a criação das sequencia que podem ser repetições de mesmo valor com soma menor que 540. Exemplo: [200] = [2, 200]

3.2.1 Exemplo de entrada e saída da função de combinações

Como exemplo de dados de entrada, foi utilizado a seguinte combinação:

3	4
10	200
5	330
10	420
8	500

Onde a primeira linha são os valores de entrada número de máquinas (m) e número de pedidos (n) respectivamente. Da segunda linha em diante a entrada tem o formato de quantidade de pedidos para aquele tempo (n_i) e valor de tempo para o pedido (t_i). A saída esperada desse conjunto de entradas após passar pela função *combinacao_valores* pode ser visto a seguir:

[1, 200, 1, 330 , [2, 200], [1, 420], [1, 500]].

Esses valores presentes na lista podem ser compreendidos da seguinte forma: Valores na posição (i) par (0,2,4, ..., 2*N) são referentes a quantidade de vezes que o elemento que o segue na posição ($i+1$) impar (1,3,5,...,2*n-1) aparecem naquele padrão.

3.3 Montagem do padrão lp_solve

Os dados fornecidos pela função de combinação são utilizados como dados de entrada para a função *criacao_lp_solve_inst*. Essa função fica responsável por pegar as combinações e transformar em uma matriz de equações, seguindo o formato de um sistema linear. Essa função segue a seguinte ordem de execução:

1. Varre a lista de combinações pegando cada t_i de pedido e montando em cada linha da matriz uma função com a quantidade de vezes que (n_i) aparece para cada t_i;
2. Verifica a quantidade de colunas geradas na matriz de equações para realizar a criação das variáveis x_i que vão fazer parte do sistema linear de saída;
3. Com a matriz de equações completa, é realizado um mapeamento das variáveis x_i em relação a cada posição de elemento de cada linha da matriz;
4. É impresso na saída padrão (*stdout*) o sistema linear no formato de entrada utilizado pela ferramenta lp_solve.
5. Simultaneamente ao passo anterior também é impresso a função objetivo do problema.

3.3.1 Exemplo de execução e saída da função de montagem

Utilizando os dados provindos pela função de combinação, teremos então na execução da primeira etapa a transformação dos dados [[1, 200, 1, 330], [2, 200], [1, 420], [1, 500]], em:

1	2	0	0
1	0	0	0
0	0	1	0
0	0	0	1

Como a segunda etapa é trivial de se entender, irei pular e ilustrar o funcionamento da terceira etapa. Utilizando a matriz gerada pela primeira etapa, então teríamos a seguinte forma:

1x0	2x1	0x2	0x3
1x0	0x1	0x2	0x3
0x0	0x1	1x2	0x3
0x0	0x1	0x2	1x3

E durante a etapa de impressão é realizado o resto do tratamento na matriz obtida na etapa anterior e a impressão da função objetivo. Esse ultimo tratamento é referente a percorrer a matriz e retirar os elementos iguais a 0 que estão multiplicando os x_i, e atribuindo o valor de inequação para cada um dos pedidos de tempo referentes as equações, tomando o formato:

min: x0 + x1 + x2 + 3;

1x0 + 2x1 >= 10;
1x0 >= 5;
1x2 >= 10;
1x3 >= 8;

4 Referencias

KNAPSACK problem. 17 jun. 2021.Disponível em:
https://en.wikipedia.org/wiki/Knapsack_problem. Acesso em: 20 jun. 2021.