Escalonamento de Tarefas em Máquinas por Programação Linear

Departamento de Informática
Universidade Federal do Paraná
Curitiba, Paraná - Brasil

Victor Picussa
vp16@inf.ufpr.br

I. Introdução

O problema do escalonamento de tarefas em máquinas distintas com custo, tem como objetivo alocar as tarefas por máquinas minimizando o makespan e, consequentemente, minimizar o custo de execução de todas as tarefas. Para entender um pouco melhor sobre o problema, o artigo [1] sobre Job Shop foi usado como base.

II. O Problema

Seja T = $\{t_1, t_2, ..., t_n\}$ um conjunto de tarefas, tal que n é o número total de tarefas. Um conjunto M = $\{m_1, m_2, ..., m_k\}$ de máquinas, tal que k é o número total de máquinas. Cada tarefa t $\in$ T possui um tempo de execução h_t. Cada máquina m $\in$ M possui um tempo de operação máximo u_m e o custo por unidade de execução c_m. O objetivo é minimizar o custo da execução de todas as tarefas.

III. Modelagem do Problema

Para modelar o problema é necessário dividir em algumas seções: definições, variáveis de decisão, função objetivo e restrições. Para a modelagem foi utilizado o livro [2] como estudo para criar o modelo a seguir.

A. Definições

- h_i := tempo de duração de execução da tarefa i
- u_j := tempo de atividade máxima da máquina j
- x_{ij} := recebe 1 se é executável na máquina j, 0 caso contrário
- c_j := custo de execução de uma tarefa por unidade de tempo na máquina j

B. Variáveis de Decisão

- t_{ij} := tempo de duração de execução da tarefa i na máquina j

C. Função Objetivo

- min $\sum_{j \in M} \sum_{i \in T} t_{ij} c_j$

D. Restrições

- $\sum_{i \in T} t_{ij} x_{ij} <= u_j$, $\forall$ j $\in$ M
- $\sum_{j \in M} t_{ij} = h_i$, $\forall$ i $\in$ T
- t_{ij} = 0, $\forall$ x_{ij} = 0, i $\in$ T, j $\in$ M
- t_{ij} >= 0, $\forall$ i $\in$ T, j $\in$ M

E. Explicação da Modelagem

A partir da função objetivo queremos minimizar o valor da multiplicação do tempo de utilização das tarefas nas respectivas máquinas t_{ij} pelo valor hora da utilização da máquina c_j. O resultado é o custo total dessa execução. Para adequar ao problema, as funções de restricão seguem:

- A soma do tempo de execução das tarefas em uma máquina *j*, vezes a validade das tarefas executarem na máquina *j*, deve ser menor ou igual ao tempo máximo de utilização da máquina *j*. Com isso temos a restrição do tempo de utilização das máquinas.
- A soma do tempo de execução de um tarefa em todas as máquinas deve ser igual ao tempo necessário para executa-lá por completo. Isso significa que uma tarefa pode ser executada em multiplas máquinas, respeitando as restrições.
- O tempo de execução de uma tarefa em certa máquina deve ser nulo caso foi especificado que a máquina não pode executar aquela tarefa.

Com essas restrições, a função objetivo nos retorna o custo minímo da execução de todas as tarefas.

IV. Implementação da Resolução

A resolução desse problema foi implementada na linguagem Python, utilizando a biblioteca PuLP com o resolvedor padrão CBC MIP Solver. Como base para implementar o problema, utilizou-se o artigo [3].

O executável principal *tarefas.py* utiliza das funções presentes no arquivo *utils.py* e *lpsolver.py*. O arquivo *utils* disponibiliza uma função para ler um arquivo de entrada e organizar os dados em variáveis para o resolvedor do problema. Esse arquivo não será apresentado aqui em detalhes por ser uma leitura simples da entrada e sua divisão em variáveis. O arquivo *lpsolver* tem a função que fará a modelagem e resolução do problema. Primeiramente definimos a variável de decisão do problema como:

```
t = {(i,j):
    plp.LpVariable(cat=plp.LpContinuous,
            lowBound=0.0, upBound= h[i],
            name="x_{0}_{1}".format(i,j))
    for i in N for j in K}
```

Será uma variável continua entre 0.0 e seu tempo máximo h_i.

Definimos o objetivo do problema:

```
objective = plp.lpSum(t[i,j]*c[j]
            for i in N
            for j in K)

model.sense = plp.LpMinimize
```

Com isso, adicionamos as restrições do problema ao modelo:

```
constraints = {j : model.addConstraint(
        plp.LpConstraint(
            e=plp.lpSum(t[i,j]*x[i,j] for i in N),
            sense=plp.LpConstraintLE,
            rhs=u[j],
            name="constraint_maxtime_{0}".format(j)
    ))
       for j in K}

constraints = {i : model.addConstraint(
    plp.LpConstraint(
         e=plp.lpSum(t[i,j] for j in K),
         sense=plp.LpConstraintEQ,
         rhs=h[i],
         name="constraint_time_{0}".format(i)))
   for i in N}

constraints = {j : model.addConstraint(
    plp.LpConstraint(
         e=t[zeros[j]],
         sense=plp.LpConstraintEQ,
         rhs=float(0),
         name="constraint_zero_{0}".format(j)))
   for j in range(0, len(zeros))}
```

Por fim, para resolver o modelo:

```
    model.solve(plp.PULP_CBC_CMD(msg=0))
```

A execução de um exemplo que contém 4 tarefas e 5 máquinas, definidos no arquivo *exemplo3.txt* na pasta de exemplo tem a seguinte saída:

$$\begin{bmatrix} 10.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 5.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.0 \\ 0.0 & 10.0 & 5.0 & 0.0 \\ 0.0 & 10.0 & 0.0 & 5.0 \end{bmatrix}$$

O problema nos retorna o valor 675.0, o qual é ótimo para o problema.

Observação: um problema visível que não foi resolvido, porém, não afeta o resultado, é que em alguns casos uma das tarefas t_{ij} fica com o valor de -0.0, que no caso deveria ser apenas 0.0.

REFERENCES

[1] David Applegate and William Cook. *A computational study of the job-shop scheduling problem*. 1991. https://doi.org/10.1287/ijoc.3.2.149.

[2] Jirí Matoušek and Bernd Gärtner. *Understanding and Using Linear Programming*. Springer, 2006.

[3] Mohsen Moarefdoost. Optimization modeling in python: Pulp, gurobi, and cplex. 2018. https://medium.com/opex-analytics/optimization-modeling-in-python-pulp-gurobi-and-cplex-83a62129807a.