CI1238: Escalonamento de Usos no Tempo

Diogo Paris Kraut - GRR20166365

Universidade Federal do Paraná

29 de junho de 2021

ESCALONAMENTO DE USOS NO TEMPO

Introdução ao Problema

Uma empresa aluga o tempo de uso de suas máquinas para seus clientes. Supondo que essa empresa possui *m* máquinas que trabalham 9 horas por dia (540 minutos), e que os pedidos são feitos em minutos e não podem ser divididos em dias diferentes, queremos determinar como escalonar esses pedidos de forma a reduzir o número de dias necessários para estes serem completados.

Método

Usaremos o método que Matoušek e Gärtner utilizaram na seção 2.7 do seu livro *Understanding and Using Linear Programming*[1]. Esse método consiste em gerar todos os padrões de escalonamento que são válidos, ou seja, não excedem a carga horária máxima de 540**m* minutos, mas também não deixam uma sobra maior que o menor pedido. Por exemplo, suponha que foram feitos os seguintes pedidos para um empresa com uma máquina (*m*=1):

- 6 pedidos de 50 minutos;
- 5 pedidos de 200 minutos;
- 7 pedidos de 300 minutos.

Os padrões válidos seriam:

P1: 1×300 + 1×200;

P2: 2×200 + 2×50;

P3: 1×300 + 4×50;

P4: 1×200 + 6×50.

Para cada padrão P_j gerado, vamos introduzir uma variável x_j que representa a quantidade de vezes que P_j foi utilizado. Como cada padrão representa um dia inteiro de trabalho, então queremos minimizar $\sum_{j=1}^{t} x_j$, onde *t* é a quantidade de padrões.

ESCALONAMENTO DE USOS NO TEMPO

Também queremos garantir que todos os pedidos sejam atendidos, portanto vamos gerar restrições para cada tamanho de pedido. Por exemplo, para satisfazer a demanda de 5 pedidos de 200 minutos teremos a restrição $x_1 + 2 \times x_2 \geq 200$. Também vamos gerar uma restrição para que nenhuma variável seja negativa.

Como os padrões garantem que os dias serão preenchidos de forma a minimizar o tempo do expediente em que não se trabalha em nenhum pedido, e as restrições garantem que todos os pedidos serão atendidos, minimizar a quantidade de padrões utilizados é o mesmo que minimizar os dias necessários para atender todos os pedidos.

Gerando os padrões de escalonamento

Para gerar os padrões de escalonamento usamos um algoritmo recursivo cuja execução pode ser visualizada como uma árvore *n*-ária (cada nó tem 0..*n* filhos), onde *n* é o número de tempos diferentes nos pedidos (i.e. 300, 200 e 50 minutos implica em $n = 3$). Cada nó opera sobre um índice de um vetor de inteiros que representam os coeficientes que multiplicam os tempos dos pedidos. Por exemplo, o vetor a seguir representa o padrão P3 do exemplo anterior

4	0	1

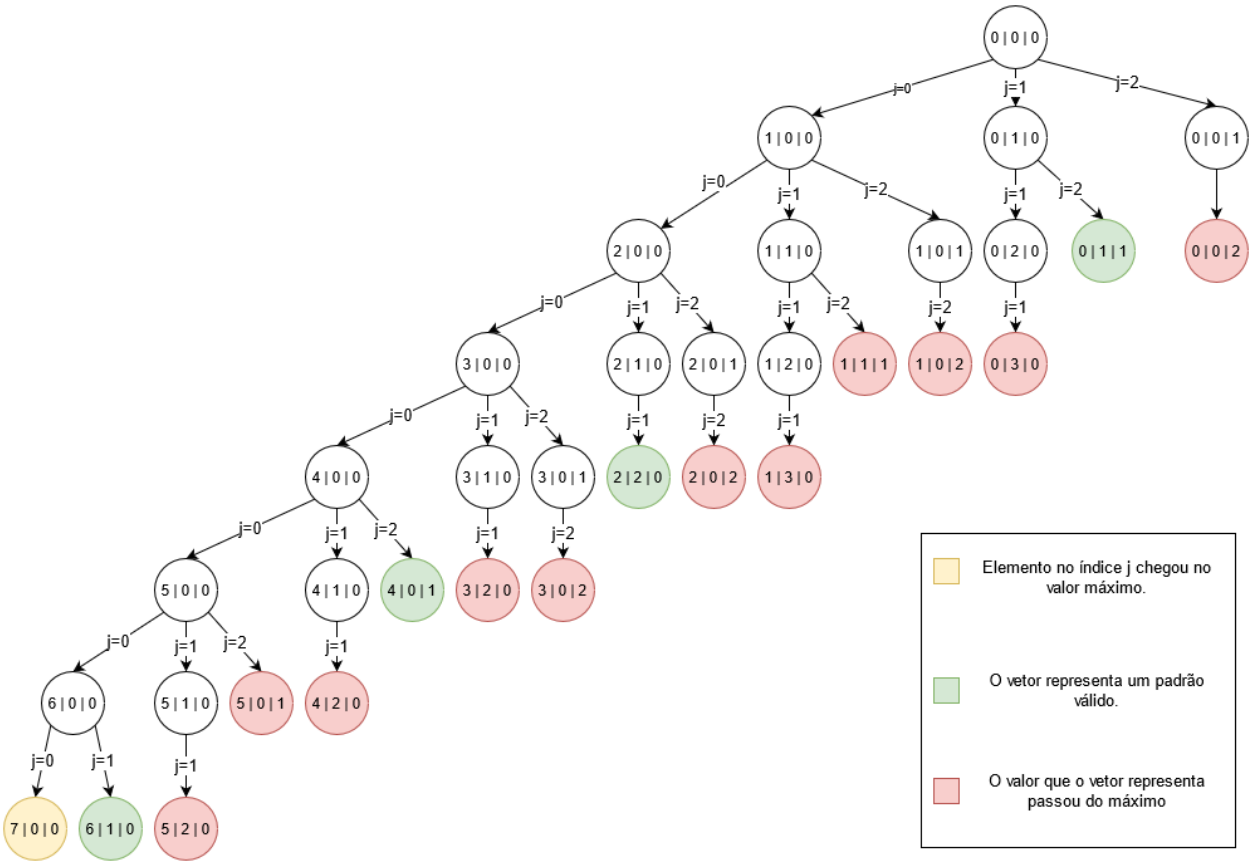
Percorremos essa árvore em profundidade seguindo as seguintes regras:

1. O primeiro filho de cada nó incrementa o valor do elemento no índice do pai.
2. Os filhos seguintes operam sobre o próximo índice de seu irmão à esquerda.
3. O valor de um elemento do vetor não pode exceder a quantidade de pedidos de sua largura.
4. Se o valor do padrão que o vetor representa exceder o tempo do expediente ($540 \times m$), então é folha e não precisamos verificar seus irmãos à direita.
5. Se o padrão que o vetor representa for válido, o nó é folha e não precisamos verificar seus irmãos à direita.

ESCALONAMENTO DE USOS NO TEMPO

Esse algoritmo tem complexidade exponencial da forma c^n no pior caso, onde c seria uma função do número total de pedidos, mas, devido às regras *3, 4 e 5*, na grande maioria dos casos complexidade é reduzida à uma função polinomial.

Exemplos Completos



Árvore de execução do exemplo dado na descrição do método.

Exemplo 1.in

Entrada:

3 4
10 200
5 330
10 420
8 500

ESCALONAMENTO DE USOS NO TEMPO

Saída:

min: +x0 +x1 +x2 +x3 +x4 +x5 +x6 +x7 +x8 +x9 +x10 +x11 +x12 +x13 +x14 +x15 +x16 +x17 +x18 +x19 +x20 +x21 +x22 +x23 +x24 +x25;
pattern_00: x0 = +8 200;
pattern_01: x1 = +6 200 +1 330;
pattern_02: x2 = +6 200 +1 420;
pattern_03: x3 = +5 200 +1 500;
pattern_04: x4 = +4 200 +2 330;
pattern_05: x5 = +4 200 +1 330 +1 420;
pattern_06: x6 = +3 200 +3 330;
pattern_07: x7 = +3 200 +1 330 +1 500;
pattern_08: x8 = +3 200 +2 420;
pattern_09: x9 = +3 200 +1 420 +1 500;
pattern_10: x10 = +3 200 +2 500;
pattern_11: x11 = +2 200 +2 330 +1 420;
pattern_12: x12 = +2 200 +2 330 +1 500;
pattern_13: x13 = +2 200 +1 330 +2 420;
pattern_14: x14 = +1 200 +4 330;
pattern_15: x15 = +1 200 +3 330 +1 420;
pattern_16: x16 = +1 200 +1 330 +1 420 +1 500;
pattern_17: x17 = +1 200 +1 330 +2 500;
pattern_18: x18 = +1 200 +3 420;
pattern_19: x19 = +1 200 +2 420 +1 500;
pattern_20: x20 = +1 200 +1 420 +2 500;
pattern_21: x21 = +3 330 +1 500;
pattern_22: x22 = +2 330 +2 420;
pattern_23: x23 = +2 330 +1 420 +1 500;
pattern_24: x24 = +1 330 +3 420;
pattern_25: x25 = +3 500;
+8 x0 +6 x1 +6 x2 +5 x3 +4 x4 +4 x5 +3 x6 +3 x7 +3 x8 +3 x9 +3 x10 +2 x11 +2 x12 +2 x13 +1 x14 +1 x15 +1 x16 +1 x17 +1 x18 +1 x19 +1 x20 >= 10;
+1 x1 +2 x4 +1 x5 +3 x6 +1 x7 +2 x11 +2 x12 +1 x13 +4 x14 +3 x15 +1 x16 +1 x17 +3 x21 +2 x22 +2 x23 +1 x24 >= 5;
+1 x2 +1 x5 +2 x8 +1 x9 +1 x11 +2 x13 +1 x15 +1 x16 +3 x18 +2 x19 +1 x20 +2 x22 +1 x23 +3 x24 >= 10;
+1 x3 +1 x7 +1 x9 +2 x10 +1 x12 +1 x16 +2 x17 +1 x19 +2 x20 +1 x21 +1 x23 +3 x25 >= 8;
x0 >= 0;
x1 >= 0;
x2 >= 0;
x3 >= 0;
x4 >= 0;
x5 >= 0;
x6 >= 0;
x7 >= 0;

x8 >= 0;
x9 >= 0;
x10 >= 0;
x11 >= 0;
x12 >= 0;
x13 >= 0;
x14 >= 0;
x15 >= 0;
x16 >= 0;
x17 >= 0;
x18 >= 0;
x19 >= 0;
x20 >= 0;
x21 >= 0;
x22 >= 0;
x23 >= 0;
x24 >= 0;
x25 >= 0;

Exemplo 2.in

Entrada:

1 3
97 135
610 108
395 93

Saida:

min: +x0 +x1 +x2 +x3 +x4 +x5 +x6 +x7 +x8 +x9 +x10 +x11 +x12 +x13 +x14 +x15;
pattern_00: x0 = +5 93;
pattern_01: x1 = +4 93 +1 108;
pattern_02: x2 = +4 93 +1 135;
pattern_03: x3 = +3 93 +2 108;
pattern_04: x4 = +3 93 +1 108 +1 135;
pattern_05: x5 = +2 93 +3 108;
pattern_06: x6 = +2 93 +2 108 +1 135;
pattern_07: x7 = +2 93 +2 135;
pattern_08: x8 = +1 93 +4 108;
pattern_09: x9 = +1 93 +1 108 +2 135;
pattern_10: x10 = +1 93 +3 135;
pattern_11: x11 = +5 108;
pattern_12: x12 = +3 108 +1 135;
pattern_13: x13 = +2 108 +2 135;
pattern_14: x14 = +1 108 +3 135;

ESCALONAMENTO DE USOS NO TEMPO

pattern_15: x15 = +4 135;
+5 x0 +4 x1 +4 x2 +3 x3 +3 x4 +2 x5 +2 x6 +2 x7 +1 x8 +1 x9 +1 x10 >= 395;
+1 x1 +2 x3 +1 x4 +3 x5 +2 x6 +4 x8 +1 x9 +5 x11 +3 x12 +2 x13 +1 x14 >= 610;
+1 x2 +1 x4 +1 x6 +2 x7 +2 x9 +3 x10 +1 x12 +2 x13 +3 x14 +4 x15 >= 97;
x0 >= 0;
x1 >= 0;
x2 >= 0;
x3 >= 0;
x4 >= 0;
x5 >= 0;
x6 >= 0;
x7 >= 0;
x8 >= 0;
x9 >= 0;
x10 >= 0;
x11 >= 0;
x12 >= 0;
x13 >= 0;
x14 >= 0;
x15 >= 0;

References

[1] Matoušek, J. e Gärtner, B.(2007). Understanding and Using Linear Programming.

GeeksForGeeks. (2021). Algoritmo do QuickSort adaptado de

https://www.geeksforgeeks.org/quick-sort/