Planejamento de Produção em Sistemas de Manufatura

João Pedro Bizzi Velho

218711

item 1

Seguindo a notação na documentação do OR-TOOLS, podemos descrever o problema da seguinte forma:

Onde cada task é representada pelo par (m,p) onde m é a máquina pela qual deve passar e p o tempo de processamento naquela máquina. As tasks estão organizadas de acordo com o Figura 1.

$$job_0 = [(1, 10), (2, 10), (0, 10)]$$
$$job_1 = [(1, 5), (0, 8), (2, 5)]$$
$$job_2 = [(2, 5), (1, 9), (0, 9)]$$
$$job_3 = [(0, 6), (2, 9), (1, 5)]$$

Com as tasks definidas devemos definir as restrições para isto, introduz-se uma nova notação $task(i, j)$ onde j é a jésima task para o job i. Com isso, temos que $t_{i,j}$ é o tempo inicial dessa task.

Como na documentação do OR-TOOLS, definimos dois tipo de restrição, a de precedência e as para evitar sobreposição. Inicialmente definidmos as de precedência:

O tempo de início da task(0,2) ou seja, da segunda task do job 0 deve ser superior ao tempo gasto pela task(0,1), ou seja, do primeira task do job 1. Extendendo essa definição para todos os jobs dados:

$$t_{0,1} + 10 <= t_{0,2}$$
$$t_{0,2} + 10 <= t_{0,3}$$
$$t_{1,1} + 5 <= t_{1,2}$$
$$t_{1,2} + 8 <= t_{1,3}$$
$$t_{2,1} + 5 <= t_{2,2}$$
$$t_{2,2} + 9 <= t_{2,3}$$
$$t_{3,1} + 6 <= t_{3,2}$$
$$t_{3,2} + 9 <= t_{3,3}$$

Com estas restrições temos as tasks de cada job espaçadas de forma individual. Porém, não impedimos que as máquinas tenham duas tasks ao mesmo tempo, é ai que entra o segundo tipo de restrição. Temos que por exemplo, a task (0,1) é realizada na mesma máquina que a task (1,1), logo, elas não podem ser realizadas de forma simultanea, portanto:

$$t_{0,1} + 10 <= t_{1,1} \text{ ou } t_{1,1} + 5 <= t_{0,1}$$

Já para o restante das tasks na máquina 1 temos:

$$t_{1,1} + 5 <= t_{2,2} \text{ ou } t_{0,1} + 10 <= t_{2,2}$$
$$t_{2,2} + 9 <= t_{3,3}$$

Temos duas restrições possíveis pois é possível agendar a task (0,1) ou (1,1) inicialmente, logo, são dois caminhos distintos possíveis.

Para a máquina 0 temos as seguintes restrições:

$$t_{3,1} + 6 <= t_{1,2}$$
$$t_{1,2} + 8 <= t_{0,3} \text{ ou } t_{1,2} + 8 <= t_{2,3}$$
$$t_{0,3} + 10 <= t_{2,3} \text{ ou } t_{2,3} + 9 <= t_{0,3}$$

Podemos fazer a task do job 0 ou do job 2 primeiro, logo são dois caminhos possíveis novamente.

Por fim, para a máquina 2, temos as seguintes restrições:

$$t_{2,1} + 5 <= t_{0,2} \text{ ou } t_{2,1} + 5 <= t_{3,2}$$
$$t_{0,2} + 10 <= t_{3,2} \text{ ou } t_{3,2} + 9 <= t_{0,2}$$
$$t_{3,2} + 9 <= t_{1,3} \text{ ou } t_{0,2} + 10 <= t_{1,3}$$

Com isto temos todas as restrições definidas para o problema. Agora temos que montar o solver com o OR-TOOLS.

item 2 + 3

Para o código utilizei o exemplo na documentação do OR-TOOLS modificando apenas para os dados da atividade e a quantidade de jobs e máquinas.

```
[ ] ! python -m pip install --upgrade --user ortools
```

```
Requirement already up-to-date: ortools in /root/.local/lib/python3.6/site-packages (8.1.8487)
Requirement already satisfied, skipping upgrade: absl-py>=0.11 in /root/.local/lib/python3.6/site-packages (from ortools) (0.11.0)
Requirement already satisfied, skipping upgrade: protobuf>=3.14.0 in /root/.local/lib/python3.6/site-packages (from ortools) (3.14.0)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python3.6/dist-packages (from absl-py>=0.11->ortools) (1.15.0)
```

```
[ ] import collections
    from ortools.sat.python import cp_model


    def MinimalJobshopSat():
        """Minimal jobshop problem."""
        #define o modelo
        model = cp_model.CpModel()

        jobs_data = [  # task = (máquina, tempo_gasto)
            [(1, 10), (2, 10), (0, 10)],  # Job0
            [(1, 5), (0, 8), (2, 5)],  # Job1
            [(2, 5), (1, 9), (0,9)],  # Job2
            [(0, 6), (2, 9), (1, 5)],  # Job3
        ]
        # contagem de máquinas
        machines_count = 1 + max(task[0] for job in jobs_data for task in job)
        all_machines = range(machines_count)
[ ]
        # computa a soma de todas as durações de forma dinâmica
        horizon = sum(task[1] for job in jobs_data for task in job)


        # nomeia a tupla que ira guardar informação da nova variável
        task_type = collections.namedtuple('task_type', 'start end interval')
        # tupla da solução
        assigned_task_type = collections.namedtuple('assigned_task_type',
                                                    'start job index duration')


        all_tasks = {}
        machine_to_intervals = collections.defaultdict(list)

        for job_id, job in enumerate(jobs_data):
            for task_id, task in enumerate(job):
                machine = task[0]
                duration = task[1]
                suffix = '_%i_%i' % (job_id, task_id)
                start_var = model.NewIntVar(0, horizon, 'start' + suffix)
                end_var = model.NewIntVar(0, horizon, 'end' + suffix)
                interval_var = model.NewIntervalVar(start_var, duration, end_var,
                                                    'interval' + suffix)
                all_tasks[job_id, task_id] = task_type(
                    start=start_var, end=end_var, interval=interval_var)
                machine_to_intervals[machine].append(interval_var)

        # Restrição para não dar overlap, sobreposição nas máquinas.
        for machine in all_machines:
            model.AddNoOverlap(machine_to_intervals[machine])

[ ]     # Restrição de precedência
        for job_id, job in enumerate(jobs_data):
            for task_id in range(len(job) - 1):
                model.Add(all_tasks[job_id, task_id +
                                    1].start >= all_tasks[job_id, task_id].end)

        # Objetivo
        obj_var = model.NewIntVar(0, horizon, 'makespan')
        model.AddMaxEquality(obj_var, [
            all_tasks[job_id, len(job) - 1].end
            for job_id, job in enumerate(jobs_data)
        ])
        model.Minimize(obj_var)

        # Resolve o modelo criado
        solver = cp_model.CpSolver()
        status = solver.Solve(model)

        if status == cp_model.OPTIMAL:
            assigned_jobs = collections.defaultdict(list)
            for job_id, job in enumerate(jobs_data):
                for task_id, task in enumerate(job):
                    machine = task[0]
                    assigned_jobs[machine].append(
                        assigned_task_type(
                            start=solver.Value(all_tasks[job_id, task_id].start),
                            job=job_id,
                            index=task_id,
                            duration=task[1]))

            # printa a saolução ótima na forma de uma tabela
            output = ''
[ ]         for machine in all_machines:
                assigned_jobs[machine].sort()
                sol_line_tasks = 'Máquina ' + str(machine) + ': '
                sol_line = '           '

                for assigned_task in assigned_jobs[machine]:
                    name = 'job_%i_%i' % (assigned_task.job, assigned_task.index)
                    sol_line_tasks += '%-10s' % name

                    start = assigned_task.start
                    duration = assigned_task.duration
                    sol_tmp = '[%i,%i]' % (start, start + duration)
                    sol_line += '%-10s' % sol_tmp

                sol_line += '\n'
                sol_line_tasks += '\n'
                output += sol_line_tasks
                output += sol_line

            print('Tempo ótimo %i' % solver.ObjectiveValue())
            print(output)


    MinimalJobshopSat()
```

```
Tempo ótimo 42
Máquina 0: job_3_0   job_1_1   job_0_2   job_2_2
           [0,6]     [15,23]   [23,33]   [33,42]
Máquina 1: job_0_0   job_1_0   job_2_1   job_3_2
           [6,16]    [16,21]   [15,24]   [29,34]
Máquina 2: job_2_0   job_0_1   job_1_1   job_1_2
           [0,5]     [16,26]   [26,29]   [29,34]
```

Acima temos a tabela com as tasks para cada máquina, e o par de número dentro dos colchetes representa o tempo de início e fim de cada task na máquina.

Plot do gráfico pedido, onde o eixo y representa as máquinas, de 0 a 2 e o eixo x representa o tempo gasto na máquina. Temos também a legenda com cada job com cores diferentes.

```
[ ] import matplotlib.pyplot as plt

    # Declaração do tipo de figura
    fig, gnt = plt.subplots()

    # Definição do tamanho dos eixos
    gnt.set_ylim(0, 50)

    gnt.set_xlim(0, 45)

    # Nomes dos eixos
    gnt.set_xlabel('Tempo decorrido')
    gnt.set_ylabel('Máquina')

    # Espacamento do eixo y
    gnt.set_yticks([15, 25, 35])
    # Marcações no eixo y
    gnt.set_yticklabels(['0', '1', '2'])


    gnt.grid(True)

[ ] # Job 0 na máquina 0
    gnt.broken_barh([(23, 10)], (10, 9), facecolors ='tab:orange',label = 'Job 0')
    # Job 1 na máquina 0
    gnt.broken_barh([(15, 8)], (10, 9), facecolors ='tab:red'),label = 'Job 1')
    # Job 2 na máquina 0
    gnt.broken_barh([(33, 9)], (10, 9), facecolors ='tab:purple'),label = 'Job 2')
    # Job 3 na máquina 0
    gnt.broken_barh([(0, 6)], (10, 9), facecolors ='tab:blue'),label = 'Job 3')

    # Job 0 na máquina 1
    gnt.broken_barh([(6, 10)], (20, 9), facecolors ='tab:orange'))
    # Job 1 na máquina 1
    gnt.broken_barh([(16, 5)], (20, 9), facecolors ='tab:red'))
    # Job 2 na máquina 1
    gnt.broken_barh([(15, 9)], (20, 9), facecolors ='tab:purple'))
    # Job 3 na máquina 1
    gnt.broken_barh([(29, 5)], (20, 9), facecolors ='tab:blue'))

    # Job 0 na máquina 2
    gnt.broken_barh([(16, 20)], (30, 9), facecolors ='tab:orange'))
    # Job 1 na máquina 2
    gnt.broken_barh([(29, 5)], (30, 9), facecolors ='tab:red'))
    # Job 2 na máquina 2
    gnt.broken_barh([(0, 5)], (30, 9), facecolors ='tab:purple'))
    # Job 3 na máquina 2
    gnt.broken_barh([(20, 9)], (30, 9), facecolors ='tab:blue'))
    print("Diagrama de Gantt da solução ótima")
    gnt.legend()
```

```
Diagrama de Gantt da solução ótima
<matplotlib.legend.Legend at 0x7fa15a3f77b0>
```