

Lógica Computacional - TP1 Exercício 1 - G01

Bruno Dias da Gíão A96544, João Luis da Cruz Pereira A95375, David Alberto Agra A95726

October 1, 2023

0.1 Exercício 1 - Enunciado

1. Pretende-se construir o horário semanal de aulas de uma turma.
 1. Existe um conjunto de salas S classificadas em “grandes” e “pequenas”.
 2. O tempo do horário está organizado em “slots” de uma hora. O total do tempo disponível é 5 horas de manhã e 5 horas à tarde.
 3. Existe um conjunto D de disciplinas. Cada disciplina tem um atributo $d \in \{1, 2\}$ que classifica a duração de cada sessão (um ou dois “slots”), um atributo $a \in \{2, 3\}$ que define o número de sessões semanais e um atributo $s \in \{0, 1\}$ que diz se a sessão necessita de uma sala grande ou não.
 - d. Existe um conjunto P de professores. Cada professor tem associado um conjunto h das disciplinas que está habilitado a lecionar.
 - e. O horário está organizado em sessões concorrentes onde cada sessão é definido por uma disciplina desce que salas e professores verifiquem as seguintes restrições.
 1. Para cada disciplina todas as aulas decorrem na mesma sala e com o mesmo professor.
 2. O número total de horas lecionadas por cada professor está num intervalo de $\pm 20\%$ do número médio de horas lecionadas pela totalidade dos professores.
 3. Nenhuma sala pode ser ocupada simultaneamente por mais do que uma aula e nenhum professor pode lecionar simultaneamente mais do que uma aula.
 4. Em cada disciplina, cada aula é lecionada por um professor habilitado para essa disciplina e ocorre numa sala de tamanho apropriado à disciplina.

Use o package `ortools` para encontrar uma solução que verifique todas as restrições e maximize o número de partes de dia (manhãs ou tardes) que estão livres de qualquer aula.

0.2 Exercício 1 - Solução

0.2.1 Exercício 1 - Restrições e Metodologia

Os inputs seguirão o seguinte formato, S, D, P sendo inputs numéricos que remetem para o tamanho dos conjuntos descritos no enunciado e gp, ph, dd, na, ts sendo arrays que contêm informação sobre as salas, docentes e disciplinas.

- gp será um array booleano onde o sinal positivo 1 representa uma sala de tamanho grande e o *bit* 0 representa uma sala pequena;
- ph será uma matriz booleana $D \times P$ onde cada coluna representa uma disciplina e as linhas representam a equipa docente, com informação sobre se cada docente está habilitado para ensinar uma dada disciplina.

- *dd* será um array com os elementos do atributo *d* descrito no enunciado, isto é, $d \in \{1, 2\}$, que será a informação sobre quantas horas cada aula de uma disciplina dura.
- *na* é um array com os elementos do atributo *a* descrito no enunciado, isto é, $a \in \{2, 3\}$, que representa quantas aulas semanais cada disciplina tem.
- *ts* é um array booleano com a informação sobre se uma dada disciplina necessita ou não de uma sala grande.

Como, por exemplo, o seguinte conjunto de *input*:

```
S = 3
D = 3
P = 3
```

```
gp = [0, 1, 1]
ph = [[0, 1, 0], [1, 1, 0], [0, 0, 1]]
dd = [2, 2, 2]
na = [2, 2, 2]
ts = [0, 1, 0]
```

De forma a encontrar solução para este problema, utilizaremos o Solver de Programação de Inteiros com restrições, SCIP, proveniente do package *ortools*

```
from ortools.linear_solver import pywraplp
```

```
solver = pywraplp.Solver.CreateSolver('SCIP')
```

Começamos por criar o dicionário onde iremos guardar a ocorrência de aulas num determinado dia, hora, sala, disciplina e professor. O solver preencherá com uma flag binária, indicando a existência de aulas num dado *slot*, através de uma *BoolVar*.

```
for dia in range(dias):
    h[dia] = {}
    for hora in range(horas_manha+horas_tarde):
        h[dia][hora] = {}
        for s in range(S):
            h[dia][hora][s] = {}
            for d in range(D):
                h[dia][hora][s][d] = {}
                for p in range(P):
                    h[dia][hora][s][d][p] = solver.BoolVar("[%i] [%i] [%i] [%i] [%i]"
                                                                % (dia, hora, s, d, p))
```

De seguida criamos dois dicionários auxiliares que irão guardar qual é a sala de cada disciplina e qual o professor de cada disciplina, conseguindo assim assegurar a unicidade de sala e professor por disciplina.

```
for s in range(S):
    sala_disc[s] = {}
    for d in range(D):
        sala_disc[s][d] = solver.BoolVar("[%i] [%i]" % (s, d))

for d in range(D):
```

```

disc_prof[d] = {}
for p in range(P):
    disc_prof[d][p] = solver.BoolVar("[%i] [%i]" % (d, p))

```

Esta restrição garante que a duração de cada aula está correta, para isso estamos a percorrer todos os parâmetros (dias, horas, salas, disciplinas, professores) e para cada disciplina verificamos qual a duração de aula pretendida (1 ou 2 horas).

Verificamos também casos base (1ª hora da manhã, última hora da manhã, 1ª hora da tarde, última hora da tarde). Fazemos isto pois no caso de a duração da aula ser de duas horas queremos verificar se há uma aula da mesma disciplina na hora anterior ou na hora seguinte. Para isso fazemos o somatório da hora anterior com a hora seguinte.

Nos restantes casos só temos que verificar a hora seguinte quando é a 1ª hora manhã/tarde ou a hora anterior quando é a última hora manhã/tarde.

```

# Duração de aula correta
for dia in range(dias):
    for hora in range(horas_manha + horas_tarde):
        for d in range(D):
            for s in range(S):
                for p in range(P):
                    if (dd[d] == 2 and hora != 0 and hora != horas_manha-1 and
                        hora != horas_manha and hora != horas_manha+horas_tarde-1):
                        solver.Add(h[dia][hora][s][d][p] <= solver.Sum(h[dia][h1][s][d][p]
                            for h1 in range(hora-1, hora+2, 2)))
                    elif (dd[d] == 2 and hora == horas_manha-1):
                        solver.Add(h[dia][hora][s][d][p] <= h[dia][hora-1][s][d][p])
                    elif (dd[d] == 2 and hora == horas_manha):
                        solver.Add(h[dia][hora][s][d][p] <= h[dia][hora+1][s][d][p])
                    elif (dd[d] == 2 and hora == 0):
                        solver.Add(h[dia][hora][s][d][p] <= h[dia][hora+1][s][d][p])
                    elif (dd[d] == 2) and hora == horas_manha+horas_tarde-1:
                        solver.Add(h[dia][hora][s][d][p] <= h[dia][hora-1][s][d][p])
                    else:
                        solver.Add(h[dia][hora][s][d][p] == 0)

```

Nesta situação, “bloqueamos” cada disciplina e de seguida percorremos todos os parâmetros para a mesma e verificamos que o somatório do número de aulas dessa disciplina é igual ao número de aulas a multiplicar pela duração da mesma.

```

# Número de horas de aulas correto
for d in range(D):
    solver.Add(solver.Sum(h[dia][hora][s][d][p] for dia in range(dias)
        for hora in range(horas_manha+horas_tarde)
        for s in range(S)
        for p in range(P)
    ) == na[d]*dd[d])

```

Para confirmar que o professor atribuído pode dar a disciplina fazemos o contrário e encontramos todas as disciplinas que o professor não está habilitado a dar a disciplina e colocamos todas as

ocorrências dessa disciplina com esse professor como 0 (Não há aula).

Adicionamos esse valor também ao dicionário auxiliar para posteriormente verificarmos que só há um professor por cada disciplina.

```
# Professor poder dar a disciplina
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    if ph[p][d] == 0:
                        solver.Add(h[dia][hora][s][d][p] == 0)
                        solver.Add(disc_prof[d][p] == 0)
```

No caso de verificar que a sala tem o tamanho adequado à disciplina, usamos um processo idêntico ao de verificar que o professor pode dar a disciplina.

Percorremos todos os parâmetros e de seguida verificamos se o tamanho da sala necessário para a disciplina é diferente do tamanho da sala.

De seguida adicionamos ao dicionário principal esse valor e também ao auxiliar para posteriormente verificarmos que só há uma sala por cada disciplina.

```
# Sala tem o tamanho certo
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for p in range(P):
            for d in range(D):
                for s in range(S):
                    if ts[d] != gp[s]:
                        solver.Add(h[dia][hora][s][d][p] == 0)
                        solver.Add(sala_disc[s][d] == 0)
```

Um professor só pode lecionar uma aula em cada *slot* de hora, então vamos confirmar que em cada dia, em cada hora e para cada professor, o número total de horas de aulas é no máximo 1.

```
# Um professor não pode lecionar duas aulas paralelas
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for p in range(P):
            solver.Add(solver.Sum(h[dia][hora][s][d][p]
                                   for d in range(D) for s in range(S)) <= 1)
```

texto

```
# Não pode haver aulas da mesma disciplina ao mesmo tempo
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for d in range(D):
            solver.Add(solver.Sum(h[dia][hora][s][d][p]
                                   for s in range(S) for p in range(P)) <= 1)
```

texto

```
# Um professor por disciplina
for d in range(D):
    solver.Add(solver.Sum(disc_prof[d][p] for p in range(P)) == 1)

for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    solver.Add(h[dia][hora][s][d][p] <= disc_prof[d][p])
```

texto

```
# A mesma sala para cada disciplina
for d in range(D):
    solver.Add(solver.Sum(sala_disc[s][d] for s in range(S)) == 1)

for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    solver.Add(h[dia][hora][s][d][p] <= sala_disc[s][d])
```

A carga horária de qualquer membro da equipa docente, tem de seguir uma dada fórmula:

Seja $Avg \in \mathbb{Q}$, na e dd os parâmetros de *input* descritos no início da presente secção, $ub, lb \in \mathbb{N}$ e dda, naa são estruturas arbitrárias que representam os *inputs* dd e na adaptados às disciplinas associadas a cada docente pelo Solver:

$$Avg := \frac{\sum_{i=0}^D (na_i \times dd_i)}{P}; lb, ub := 0.8 \times Avg, 1.2 \times Avg$$

Temos que:

$$\forall_{i=0}^P (lb \leq \sum_{j=0}^{\#dda_i} ((dda_i)_j \times (naa_i)_j) \leq ub)$$

Tem de se verificar.

Para tal efeito, tratamos de calcular a média de cargos horários por todos os professores *a priori* e finalmente aplicar a fórmula, embora, em vez de utilizar estruturas auxiliares, aproveitamos o dicionário anterior *disc_prof* para calcular ou o número de horas de cada docente, ou se não existem essas horas, aproveitando o facto de *disc_prof* ser uma variável booleana.

```
# Número de horas lecionadas de +/-20%
total = 0
for i in range(D):
    total += na[d]*dd[d]
media = total / P
```

```

lower_bound = round(media*0.8)
upper_bound = round(media*1.2)

for p in range(P):
    solver.Add(lower_bound <= solver.Sum(disc_prof[d][p] * na[d] * dd[d]
                                           for d in range(D))
               <= upper_bound)

```

Texto

```

# Maximizar partes de dia livre
sum_manha = [solver.IntVar(0, 1, 'sum_manha') for dia in range(dias)]
sum_tarde = [solver.IntVar(0, 1, 'sum_tarde') for dia in range(dias)]
sum_dias = [solver.IntVar(0, 1, 'sum_dias') for dia in range(dias)]

for dia in range(dias):
    for hora in range(horas_manha):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    sum_manha[dia] += h[dia][hora][s][d][p]
                    sum_dias[dia] += h[dia][hora][s][d][p]
    for hora in range(horas_manha, horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    sum_tarde[dia] += h[dia][hora][s][d][p]
                    sum_dias[dia] += h[dia][hora][s][d][p]
solver.Maximize(solver.Sum(sum_dias) + solver.Sum(sum_manha)*0.1)

```

0.2.2 Exercício 1 - Testes

Primeiramente, teremos o exemplo simples (demonstrado no início do relatório) que consiste num input generico, onde:

$$S := 3; D := 3; P := 3$$

Especificamente:

$$gp := [0, 1, 1]; ph := [[0, 1, 0], [1, 1, 0], [0, 0, 1]]; dd := [2, 2, 2]; na := [2, 2, 2]; ts := [0, 1, 0]$$

```

[ ]: S = 3
     D = 3
     P = 3

     gp = [0, 1, 1]
     ph = [[0, 1, 0], [1, 1, 0], [0, 0, 1]]
     dd = [2, 2, 2]
     na = [2, 2, 2]
     ts = [0, 1, 0]

```

Consideremos agora o seguinte conjunto de inputs cujo solver interpreta como um problema *infeasible*.

Seja $Avg \in \mathbb{Q}$:

$$dd_0, na_0 := 1, 2; dd_1, na_1 := 2, 2; dd_2, na_2 := 2, 3; P := 3; D := 3;$$

$$Avg := \frac{\sum_{i=0}^D (dd_i \times na_i)}{P}$$

Sendo assim, o resultado de Avg será 4.

Seja dd_atrib e na_atrib o conjunto com a duração e número de aulas de cada disciplina atribuída a cada docente pelo Solver, pela fórmula anterior:

$$\forall_{i=0}^P (3 \leq \sum_{j=0}^{\#dda_i} ((dda_i)_j \times (naa_i)_j) \leq 5)$$

Trivialmente, uma das disciplinas (onde $i := 2$) terá uma duração semanal de 6 horas, sendo assim, este problema nunca será possível com esta restrição:

```
[ ]: S = 3
      D = 3
      P = 3

      gp = [0, 1, 1]
      ph = [[0, 1, 0], [1, 1, 0], [0, 0, 1]]
      dd = [1, 2, 2]
      na = [2, 2, 3]
      ts = [0, 1, 0]
```

Os seguintes dois exemplos são inputs com solução encontrada pelo Solver, a sua criação segue o mesmo padrão que os anteriores.

```
[ ]: S = 3
      D = 4
      P = 4

      gp = [0, 1, 1]
      ph = [[0, 1, 1, 0], [1, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 0]]
      dd = [2, 2, 2, 2]
      na = [3, 2, 2, 3]
      ts = [0, 1, 1, 0]
```

```
[ ]: S = 3
      D = 4
      P = 4

      gp = [0, 1, 1]
      ph = [[0, 1, 1, 0], [1, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 0]]
```

```
dd = [2, 2, 2, 2]
na = [2, 2, 2, 2]
ts = [0, 1, 1, 0]
```

0.3 Exercício 1 - Anexo

```
[ ]: from ortools.linear_solver import pywraplp
solver = pywraplp.Solver.CreateSolver('SCIP')

dias = 5
horas_manha = 5
horas_tarde = 5
h = {}

sala_disc = {}
disc_prof = {}

for dia in range(dias):
    h[dia] = {}
    for hora in range(horas_manha+horas_tarde):
        h[dia][hora] = {}
        for s in range(S):
            h[dia][hora][s] = {}
            for d in range(D):
                h[dia][hora][s][d] = {}
                for p in range(P):
                    h[dia][hora][s][d][p] = solver.
                    ⇨ BoolVar("[%i] [%i] [%i] [%i] [%i]" % (dia, hora, s, d,
                    ⇨ p))

for s in range(S):
    sala_disc[s] = {}
    for d in range(D):
        sala_disc[s][d] = solver.BoolVar("[%i] [%i]" % (s, d))

for d in range(D):
    disc_prof[d] = {}
    for p in range(P):
        disc_prof[d][p] = solver.BoolVar("[%i] [%i]" % (d, p))

# Duração de aula correta
for dia in range(dias):
    for hora in range(horas_manha + horas_tarde):
        for d in range(D):
            for s in range(S):
                for p in range(P):
```



```

        if (dd[d] == 2 and hora != 0 and hora != horas_manha-1 and
            hora != horas_manha and hora !=
↳horas_manha+horas_tarde-1):
            solver.Add(h[dia][hora][s][d][p]
                        <= solver.Sum(h[dia][h1][s][d][p]
                                       for h1 in
                                       range(hora-1, hora+2, 2)))
        elif (dd[d] == 2 and hora == horas_manha-1):
            solver.Add(h[dia][hora][s][d][p] <=
↳h[dia][hora-1][s][d][p])
        elif (dd[d] == 2 and hora == horas_manha):
            solver.Add(h[dia][hora][s][d][p] <=
↳h[dia][hora+1][s][d][p])
        elif (dd[d] == 2 and hora == 0):
            solver.Add(h[dia][hora][s][d][p] <=
↳h[dia][hora+1][s][d][p])
        elif (dd[d] == 2 and hora == horas_manha+horas_tarde-1:
            solver.Add(h[dia][hora][s][d][p] <=
↳h[dia][hora-1][s][d][p])
        else:
            solver.Add(h[dia][hora][s][d][p] == 0)

# Número de horas de aulas correto
for d in range(D):
    solver.Add(solver.Sum(h[dia][hora][s][d][p] for dia in range(dias)
                          for hora in
↳range(horas_manha+horas_tarde)
                          for s in range(S)
                          for p in range(P)
                          ) == na[d]*dd[d])

# Professor poder dar a disciplina
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    if ph[p][d] == 0:
                        solver.Add(h[dia][hora][s][d][p] == 0)
                        solver.Add(disc_prof[d][p] == 0)

# Sala tem o tamanho certo
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):

```

```

        for p in range(P):
            for d in range(D):
                for s in range(S):
                    if ts[d] != gp[s]:
                        solver.Add(h[dia][hora][s][d][p] == 0)
                        solver.Add(sala_disc[s][d] == 0)

# Uma aula por hora por professor
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for p in range(P):
            solver.Add(solver.Sum(h[dia][hora][s][d][p]
                                   for d in range(D)
                                   for s in range(S)) <= 1)

# Não pode haver aulas da mesma disciplina ao mesmo tempo
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for d in range(D):
            solver.Add(solver.Sum(h[dia][hora][s][d][p]
                                   for s in range(S)
                                   for p in range(P)) <= 1)

# Uma sala não pode estar ocupada por mais que uma disciplina na mesma hora
for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for s in range(S):
            solver.Add(solver.Sum(h[dia][hora][s][d][p]
                                   for d in range(D)
                                   for p in range(P)) <= 1)

# Um professor por disciplina
for d in range(D):
    solver.Add(solver.Sum(disc_prof[d][p]
                           for p in range(P)) == 1)

for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    solver.Add(h[dia][hora][s][d][p] <= disc_prof[d][p])

# A mesma sala para cada disciplina
for d in range(D):
    solver.Add(solver.Sum(sala_disc[s][d]
                           for s in range(S)) == 1)

```

```

for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    solver.Add(h[dia][hora][s][d][p] <= sala_disc[s][d])

# Número de horas lecionadas de +/-20%
total = 0
for i in range(D):
    total += na[d]*dd[d]
media = total / P
lower_bound = round(media*0.8)
upper_bound = round(media*1.2)

for p in range(P):
    solver.Add(lower_bound <= solver.Sum(disc_prof[d][p] * na[d] * dd[d]
                                           for d in range(D))
               <= upper_bound)

"""
# Maximizar partes de dia livre
sum_manha = [solver.IntVar(0, 1, 'sum_manha') for dia in range(dias)]
sum_tarde = [solver.IntVar(0, 1, 'sum_tarde') for dia in range(dias)]
sum_dias = [solver.IntVar(0, 1, 'sum_dias') for dia in range(dias)]

for dia in range(dias):
    for hora in range(horas_manha):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    sum_manha[dia] += h[dia][hora][s][d][p]
                    sum_dias[dia] += h[dia][hora][s][d][p]
    for hora in range(horas_manha, horas_manha+horas_tarde):
        for s in range(S):
            for d in range(D):
                for p in range(P):
                    sum_tarde[dia] += h[dia][hora][s][d][p]
                    sum_dias[dia] += h[dia][hora][s][d][p]
solver.Maximize(solver.Sum(sum_dias) + solver.Sum(sum_manha)*0.1)
"""

# Maximizar partes de dia livre
sum_hora = [[solver.IntVar(0, D, 'sum_hora')
              for hora in range(horas_manha+horas_tarde)]
             for dia in range(dias)]

```

```

for dia in range(dias):
    for hora in range(horas_manha+horas_tarde):
        sum_hora[dia][hora] = solver.Sum(h[dia][hora][s][d][p]
                                           for s in range(S)
                                           for d in range(D)
                                           for p in range(P))

solver.Maximize(solver.Sum(sum_hora[dia][hora]
                           for dia in range(dias)
                           for hora in range(horas_manha+horas_tarde)))

def print_horario():
    # Create a 5D list to store the schedule
    schedule = [[[[[0 for _ in range(P)] for _ in range(D)]
                  for _ in range(S)]
                for _ in range(horas_manha + horas_tarde)]
               for _ in range(dias)]

    # Extract the solution from the solver and populate the schedule
    for dia in range(dias):
        for hora in range(horas_manha + horas_tarde):
            for s in range(S):
                for d in range(D):
                    for p in range(P):
                        value = h[dia][hora][s][d][p].solution_value()
                        if value == 1:
                            schedule[dia][hora][s][d][p] = value

    # Print the schedule in a human-readable format
    for dia in range(dias):
        print(f"Day {dia + 1}:")
        for hora in range(horas_manha + horas_tarde):
            print(f"Hour {hora + 1}:", end=" ")
            for s in range(S):
                for d in range(D):
                    for p in range(P):
                        if h[dia][hora][s][d][p].solution_value() == 1:
                            print(f"Room {s + 1}, Discipline {d + 1}, Professor_
↪{p + 1}")
            print("\n")
        print() # Separate days with an empty line

r = solver.Solve()

if r == pywraplp.Solver.OPTIMAL:
    print("Solução encontrada")
    print_horario()

```

```
elif r == pywraplp.Solver.INFEASIBLE:  
    print("Infeasible")  
else:  
    print("Não foi encontrada solução")
```

```
[ ]:
```