Modelado prescriptivo para la optimización de horarios

Francisco Espiga
Agosto, 2021

Más sobre ciencia de datos: **cienciadedatos.net** (https://cienciadedatos.net)

1 Introducción

Septiembre está a la vuelta de la esquina y con él el regreso a las aulas. Colegios, institutos, universidades... todos tienen algo en común, la necesidad de encajar los horarios de las distintas asignaturas, tener en cuenta las restricciones del profesorado, aulas y otras preferencias.

¿No sería maravilloso si pudiésemos resolver este tedioso *Sudoku* automáticamente de algún modo? En este artículo vamos a explorar cómo conseguirlo mediante un modelo MIP (*Mixed Integer Programming*) que modelizaremos con la librería *pyomo* en Python y resolveremos con el *solver* CBC.

2 Preparando el entorno

Mi recomendación a la hora de instalar la librería y el solver es que uséis pip y conda por comodidad. Sin duda hay otros modos de hacerlo, pero instalar CBC desde conda facilita que pyomo localice el path donde está instalado y proporciona una integración "sin costuras".

In [1]:
```
#!pip install pyomo
#!conda install -c conda-forge coincbc
```

3 Modelado del problema

3.1 Planteamiento

Si pensamos en un semanario académico, es una tabla en la que para cada día de la semana y hora, debemos asignarle una asignatura. Además, la variable de decisión que utilizaremos será de tipo binario, esto es,

$$vbSubjectSchedule[d, h, s] = 1$$

si en ese día d y hora h, se imparte la asignatura s. En cualquier otro caso será 0.

Además, esto origina 3 conjuntos diferentes que usaremos en nuestro problema:

- $sDays$: los días de la semana para los que queremos generar nuestro horario.
- $sHours$: horas del día en las que impartimos la asignatura.
- $sSubjects$: asignaturas que tenemos que asignar.

Podemos ir empezando a construir nuestro modelo de Pyomo. En la siguiente celda se tiene un ejemplo preparado para poder ir probando que el modelo funciona a medida que se vayan implementando nuevos parámetros y restricciones. Adicionalmente definimos varios parámetros:

- $max\_hours\_per\_day$: número máximo de horas de una asignatura que se pueden impartir en un día.
- $hours\_per\_subject$: horas semanales de cada asignatura.
- $preferences$: preferencias del profesorado para impartir una asignatura un (día, hora) concreto y su peso, en forma de tupla.
- $constraints$: idéntico a preferencias, pero en este caso como restricciones de obligado cumplimiento.

In [2]:
```
import pyomo.environ as pyo
import numpy as np
import pyomo.kernel as pmo
import pandas as pd
```

In [3]:
```
# Args
days = ['l','m','x','j', 'v']
hours = [f"h_{i}" for i in np.arange(10,14,1)]
subjects = [f"SB_{i}" for i in range(8)]
hours_per_subject = dict(zip(subjects, [2 for i in range(8)]))
for i in np.random.choice(subjects, 4):
    hours_per_subject[i]+=1

max_hours_per_day = 2

preferences = [
    ('l', f"h_{11}", 'SB_3', 4)
]

constraints = [
    ('l', f"h_{12}", 'SB_1',  1)
]

hours_per_subject
```

Out[3]:
```
{'SB_0': 2,
 'SB_1': 3,
 'SB_2': 2,
 'SB_3': 2,
 'SB_4': 3,
 'SB_5': 3,
 'SB_6': 2,
 'SB_7': 3}
```

In [4]:
```
# Definición del modelo
# ----------------------------------------------------------------------------
model = pyo.ConcreteModel()

# Sets
model.sDays = pyo.Set(initialize = days, ordered = True)
model.sHours = pyo.Set(initialize = hours, ordered = True)
model.sSubjects = pyo.Set(initialize = subjects)

# Decision variable
model.vbSubjectSchedule = pyo.Var(
                            model.sDays,
                            model.sHours,
                            model.sSubjects,
                            domain = pmo.Binary
                           )
```

Ahora, es momento de comenzar a definir nuestro calendario, añadiendo al modelo distintos parámetros y variables auxiliares que nos sirvan para modelar restricciones o tener acceso rápidamente a valores una vez resuelto el modelo.
De ahora en adelante, usaremos el término **parámetros** para referirnos a cualquier componente que tome un valor definido y fijo y **variables** a aquellos componentes cuyo valor dependerá de las decisiones que tomemos.

3.2 Parámetros

- $pHoursPerSubject_{sSubjects}$: horas semanales de cada asignatura
- $pMinDaysPerSubject_{sSubjects}$: mínimo de días en los que impartir una asignatura, teniendo en cuenta el total semanal de horas y el máximo diario que se puede impartir.
- $pMaxDaysPerSubject_{sSubjects}$: en nuestro caso, tantos días como horas lectivas haya, pero en otros casos podríamos querer limitarlo.
- $pPreferences_{sDays,\ sHours,\ sSubjects}$: preferencias del profesorado. Por defecto cada asignación tendrá el mismo valor unitario, pero podremos favorecer determinadas decisiones modificando el peso de esa preferencia (d,h,s).

3.3 Variables auxiliares

- $vbSubjectDaysFlags_{sDays,sSubjects}$: variable indicativa para controlar si un día de la semana se imparte una asignatura determinada.
- $vIcumulatedHours_{sDays,\ sHours,\ sSubjects}$: variable entera con el acumulado de horas de una asignatura a lo largo de un día, lo usaremos para controlar los bloques lectivos.
- $vbSubjectSwitches_{sDays,\ sHours,\ sSubjects}$: variable indicativa que tomará el valor 1 si en la hora siguiente se imparte la asignatura y en esta no o inversamente y 0 si coinciden ambas asignaciones.
- $vIsubjectTotalDays$: variable agregadora del número total de días en que se imparte cada asignatura, por si queremos favorecer el condensar asignaturas, dentro de las restricciones, en pocos días.

In [5]:
```
# Parameters
model.pHoursPerSubject = pyo.Param(model.sSubjects, initialize = hours_per_subject)
model.pMinDaysPerSubject = pyo.Param(
                            model.sSubjects,
                            initialize = dict(zip(
                                                hours_per_subject.keys(),
                                                [round(i/max_hours_per_day) for i in list(hours_per_subject.values
())]
                                               )
                                             )
                            )

model.pMaxDaysPerSubject = pyo.Param(model.sSubjects, initialize = hours_per_subject)

model.pPreferences = pyo.Param(
    model.sDays,
    model.sHours,
    model.sSubjects,
    initialize = 1.0,
    mutable = True
)
```

In [6]:
```
# Helper variables
model.vbSubjectDaysFlags = pyo.Var(model.sDays, model.sSubjects, domain = pmo.Binary)
model.vIcumulatedHours = pyo.Var(model.sDays, model.sHours, model.sSubjects, domain = pyo.NonNegativeIntegers)
model.vIsubjectTotalDays = pyo.Var(domain = pyo.NonNegativeIntegers)
model.vbSubjectSwitches = pyo.Var(model.sDays, model.sHours, model.sSubjects, domain = pmo.Binary)
```

3.4 Restricciones

Ahora llega el momento de trasladar lo que queremos restringir en nuestra decisión al modelo matemático.

3.4.1 c1 - Asignación única

La primera restricción es que cada hora de cada día, solamente pueda impartirse una asignatura. Matemáticamente:

$$\forall_{i \in sDays} \forall_{j \in sHours} \sum_{k \in sSubjects} vbSubjectSchedule_{i,j,k} \leq 1$$

Cabe destacar que el ≤ 1 y no la igualdad estricta permite resolver casos en los que el total de horas a impartir sea inferior al total de horas asignables.

In [7]:
```
# Constraints
#-------- single assignment constraints
# :: Only one subject can be held in the classroom at the same time
model.ctOnlyOneSubject = pyo.ConstraintList()
for i in model.sDays:
    for j in model.sHours:
        model.ctOnlyOneSubject.add(sum(model.vbSubjectSchedule[i,j,k] for k in model.sSubjects)<=1)
```

3.4.2 c2 - Impartir exactamente el número de horas lectivas de cada asignatura.

En este caso, bastará con hacer que el total de asignaciones para cada asignatura sea igual estrictamente a las horas lectivas semanales.

$$\sum_{i \in sDays} \sum_{j \in sHours} \sum_{k \in sSubjects} vbSubjectSchedule_{i,j,k} = pHoursPerSubject_k$$

In [8]:
```
# :: All the scheduled hours for a subject must be exactly the total weekly number of hours
model.ctCoverAllHours = pyo.ConstraintList()
for k in model.sSubjects:
    model.ctCoverAllHours.add(
        sum(model.vbSubjectSchedule[i,j,k] for i in model.sDays for j in model.sHours)<=model.pHoursPerSubject[k]
    )

    model.ctCoverAllHours.add(
        sum(model.vbSubjectSchedule[i,j,k] for i in model.sDays for j in model.sHours)>=model.pHoursPerSubject[k]
    )
```

3.4.3 c3 - Respetar el máximo de horas lectivas por asignatura diario

En este caso, para cada día, el total de horas de cada asignatura tendrá que ser menor o igual al máximo diario.

$$\forall_{i \in sDays} \forall_{k \in sSubjects} \sum_{j \in sHours} vbSubjectSchedule_{i,j,k} <= max\_hours\_per\_day$$

In [9]:
```
# :: the total hours per day must be lower than the max total
model.ctMaxDailyHours = pyo.ConstraintList()
for k in model.sSubjects:
    for i in model.sDays:
        model.ctMaxDailyHours.add(
            sum(model.vbSubjectSchedule[i,j,k] for j in model.sHours)<=max_hours_per_day
        )
```

3.4.4 c4 - Restricción de apoyo para la variable indicativa

Esta restricción nos sirve para que la variable indicativa $vbSubjectDaysFlags$ se active en caso de que dicha asignatura se imparta ese día concreto

In [10]:
```
# :: for each subject and day, at most there can be the max daily hours alloted for that subject
model.ctSubjectDaysFlags = pyo.ConstraintList()
for k in model.sSubjects:
    for i in model.sDays:
        model.ctSubjectDaysFlags.add(
            max_hours_per_day*model.vbSubjectDaysFlags[i,k]>=sum(model.vbSubjectSchedule[i,j,k] for j in model.sHou
rs)
        )
```

3.4.5 c5 - Máximo y mínimo número de días lectivos por asignatura

Sirviéndonos de la variable indicativa que controlamos con la restricción anterior, vamos a obligar al modelo a que cada asignatura se tenga que impartir en un rango de días $[pMinDaysPerSubject, pMaxDaysPerSubject]$. En el caso que nos ocupa, es redundante esta restricción ya que podríamos impartir solo 1 hora por día de cada asignatura, pero así podríamos modificar el modelo rápidamente para considerar otras casuísticas.

In [11]:
```
# :: Each subject can be assigned to at most hours/max hours DAYS and at least 1 hour on each of the days it has be
en scheduled.
model.ctSubjectDays = pyo.ConstraintList()
for k in model.sSubjects:
    model.ctSubjectDays.add(
        sum(model.vbSubjectDaysFlags[i,k] for i in model.sDays)<=model.pMaxDaysPerSubject[k]
    )

    model.ctSubjectDays.add(
        sum(model.vbSubjectDaysFlags[i,k] for i in model.sDays)>=model.pMinDaysPerSubject[k]
    )
```

3.4.6 c6 - Asignaciones en bloques horarios consecutivos

Es importante que las asignaturas se impartan en bloques consecutivos, ya que no tendría mucho sentido impartir 1h de Cálculo, luego una de Física y de nuevo una de Cálculo. Para ello, nos vamos a servir de distintos artificios para poder modelar la restricción correctamente.

- Excepto que se pueda impartir una asignatura en la totalidad de un día, nunca podrá estar asignada la misma asignatura al final y al principio del día. Esto se consigue con la restricción

$$\forall_{i \in sDays} \forall_{k \in sSubjects} vbSubjectSchedule_{i,sHours.first,k} + vbSubjectSchedule_{i,sHours.last,k} <= 1$$

- Por otro lado, la variable $vbSubjectSwitches_{d,h,s}$ tiene que tomar el valor absoluto de la diferencia $vbSubjectSchedule_{d,h,s} - vbSubjectSchedule_{d,h-1,s}$. Esto lo conseguimos desdoblando el valor absoluto en dos restricciones:

$$\forall_{i \in sDays}\ , \forall_{j \in sHours}\ , \forall_{k \in sSubjects}\ : vbSubjectSchedule_{i,j,k} - vbSubjectSchedule_{i,j-1,k} <= vbSubjectSwitches_{i,j,k}$$

$$\forall_{i \in sDays}\ , \forall_{j \in sHours}\ , \forall_{k \in sSubjects}\ : -vbSubjectSchedule_{i,j,k} + vbSubjectSchedule_{i,j-1,k} <= vbSubjectSwitches_{i,j,k}$$

Cabe destacar que el set $sHours$ es un set ordenado y circular, por lo que calcularemos la diferencia entre cada hora y su precedente, excepto para la última hora del día que obtendremos su diferencia con la primera hora. Esto nos permite contemplar tanto la casuística de que una asignatura se imparta a mitad del día como al principio o final. Ahora, si tomamos en consideración cualquier asignatura, en un día en el que se imparte, el número de cambios (*switches*) será de 2, la diferencia entre la primera hora en la que se imparte y la previa, y análogo para la hora posterior a la de finalización y la última de la clase. Lo vemos con un diagrama:

- Asignación: [0,0,1,1,1,0]
- Diferencias absolutas: [0,1,0,0,1,0]
- Suma: 2

Por lo tanto, en los días en que se imparte la asignatura, que recordemos se capturaban gracias a la variable $vbSubjectDaysFlags$:

$$\forall_{i \in sDays}\ , \forall_{k \in sSubjects}\ : \sum_{j \in sHours} vbSubjectSwitches_{i,j,k} <= 2 * vbSubjectDaysFlags_{i,k}$$

In [12]:
```
# :: Each subject must be given in consecutive blocks
model.ctSubjectSwitches = pyo.ConstraintList()
for k in model.sSubjects:
    for i in model.sDays:
        for j in model.sHours:
            model.ctSubjectSwitches.add(
                expr = model.vbSubjectSchedule[i,j,k] - model.vbSubjectSchedule[i,model.sHours.prevw(j), k] <= mode
l.vbSubjectSwitches[i,j,k]
            )
            model.ctSubjectSwitches.add(
                expr = -model.vbSubjectSchedule[i,j,k] + model.vbSubjectSchedule[i,model.sHours.prevw(j), k] <= mod
el.vbSubjectSwitches[i,j,k]
            )

        model.ctSubjectSwitches.add(
            expr = sum(model.vbSubjectSwitches[i,j,k] for j in model.sHours)==2*model.vbSubjectDaysFlags[i,k]
        )

        if max_hours_per_day < len(model.sHours):
            model.ctSubjectSwitches.add(
                expr = model.vbSubjectSchedule[i,model.sHours.first(), k] + model.vbSubjectSchedule[i,model.sHours.
last(), k] <= 1
            )
```

3.4.7 c7 - Opcional: intentar dar las asignaturas en el menor número de días.

Habitualmente, nos interesará impartir las asignaturas en el menor número de días posibles, pero sin querer que sea una restricción dura, para permitir encontrar una solución al modelo si el encaje fuese complicado. Podemos favorecer este comportamiento penalizando el número total de días "en exceso" en los que tenemos asiganturas asignadas, respecto al caso ideal, donde cada asignatura está asignada al menor número de días posible (redondeo del número de horas lectivas entre el máximo lectivo diario).

Esta penalización se integrará en la función objetivo mediante la variable agregadora $vIsubjectTotalDays$ y le daremos un peso arbitrario de 5 ud. por cada día en exceso consumido.

$$vIsubjectTotalDays = \sum_{k}(\sum_{i} vbSubjectDaysFlags_{i,k} - pMinDaysPerSubject_{k})$$

In [13]:
```
#---------- assignment constraints
# :: try to chunk subjects in as few days as possible, penalizing additional days
model.ctCumulativeHours = pyo.ConstraintList()
model.ctCumulativeHours.add(
    model.vIsubjectTotalDays == sum(model.vbSubjectDaysFlags[i,k] for i in model.sDays for k in model.sSubjects)-su
m(model.pMinDaysPerSubject[k] for k in model.sSubjects)
)

penalty = -5
```

3.4.8 c8 - Restricciones y preferencias de asignación

Es habitual encontrarnos que el profesorado prefiere impartir clase unos días frente a otros, o incluso en el caso más drástico la imposibilidad determinados días de la semana. Para esto y de manera opcional, podemos bien favorecer algunas asignaciones (preferencias) bien bloquearlas (tanto para asignar como para no). En este caso, podemos forzar que determinadas combinaciones (día, hora, asignatura) sean 1 o 0.

In [14]:
```
for k in preferences:
    model.pPreferences[k[0],k[1],k[2]]=k[3]

model.ctFixedSlots = pyo.ConstraintList()
for k in constraints:
    v = k[3]
    if v==1:
        model.ctFixedSlots.add(expr = model.vbSubjectSchedule[k[0],k[1],k[2]]==v)
```

3.5 Función objetivo

El modelo de optimización, a parte de intentar cumplir todas las restricciones, necesita poder comparar las soluciones ante diversas decisiones hasta encontrar la óptima. Para ello, valoraremos nuestra asignación del semanario como la suma de las preferencias asignadas (con un valor por defecto de 1 para cualquier asignación) y penalizaremos cada día en exceso utilizado respecto al mínimo necesario para impartir cada asignatura. Formalmente:

$$max\ z = \sum_{i\in\ sDays}\ \sum_{j\in\ sHours}\ \sum_{i\in\ sSubjects} pPreferences_{i,j,k} * vbSubjectSchedule_{i,j,k} - penalty * vIsubjectTotalDays$$

In [15]:
```
# Objective function
maximize = 1
model.objSchedule= pyo.Objective(
    sense = -maximize,
    expr  = penalty*(model.vIsubjectTotalDays) + sum(model.pPreferences[i,j,k]*model.vbSubjectSchedule[i,j,k] for i
in model.sDays for j in model.sHours for k in model.sSubjects)
)
```

3.6 Resolución

Una vez modelado el problema con *pyomo*, es momento de enviarlo al solver y ver si encuentra una solución factible. Si nuestro modelado es correcto, tendríamos que encontrarnos con que el lunes a las 12 se imparte la asignatura SB1 y que, de ser posible, la asignatura SB3 se impartirá el mismo día a las 11.

Por otro lado, tendremos que comprobar que se cumplan todas las asignaciones de horario y que no haya asignaturas asignadas un mismo día a bloques horarios no consecutivos.

In [16]:
```
opt = pyo.SolverFactory('cbc')
res = opt.solve(model)
print(res)
```

```
Problem:
- Name: unknown
  Lower bound: 23.0
  Upper bound: 23.0
  Number of objectives: 1
  Number of constraints: 516
  Number of variables: 353
  Number of binary variables: 360
  Number of integer variables: 361
  Number of nonzeros: 153
  Sense: maximize
Solver:
- Status: ok
  User time: -1.0
  System time: 0.18
  Wallclock time: 0.2
  Termination condition: optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an optimal solution is availabl
e.
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
    Black box:
      Number of iterations: 0
  Error rc: 0
  Time: 0.20824551582336426
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

In [17]:
```
def print_schedule(model):

    hours = []
    subjects = []
    days = []
    bool_class = []
    for i in model.sDays:
        for j in model.sHours:
            for k in model.sSubjects:
                hours.append(j)
                days.append(i)
                subjects.append(k)
                bool_class.append(model.vbSubjectSchedule[i,j,k].value)

    df_schedule = pd.DataFrame({'day':days, 'hour':hours, 'subject':subjects, 'class':bool_class})
    df_schedule = df_schedule[df_schedule['class']>0].pivot(index = 'hour', columns = 'day', values = 'subject')

    return df_schedule.loc[[h for h in model.sHours] , [d for d in model.sDays]]
```

In [20]:
```
print_schedule(model)
```

Out[20]:

day	l	m	x	j	v
hour					
h_10	SB_3	SB_4	SB_6	SB_0	SB_1
h_11	SB_3	SB_4	SB_6	SB_0	SB_1
h_12	SB_1	SB_5	SB_7	SB_5	SB_2
h_13	SB_4	SB_5	SB_7	SB_7	SB_2

4 Comentarios y conclusiones

Podemos comprobar que se han tenido en consideración tanto las restricciones como las preferencias anteriormente mencionadas y que en ningún caso existen bloques no consecutivos asignados para la misma asignatura en el mismo día. En este artículo hemos visto como, de manera sencilla, podemos implementar usando Python un modelo de optimización capaz de proporcionar unos horarios que satisfagan tanto restricciones logísticas como preferencias personales.

Este modelo es una base lo suficientemente flexible como para poder expandirlo para cualquier granularidad temporal o casuística particular.

El código del modelo apificado y en un contenedor Docker pueden encontrarse en este repositorio (https://github.com/franespiga/scheduler/tree/master)

5 Información de sesión

In [21]:
```
import session_info
session_info.show(html=False)
```

```
-----
numpy       1.19.5
pandas      1.0.5
pyomo       5.7.1
sinfo       0.3.1
-----
IPython             7.12.0
jupyter_client      6.1.5
jupyter_core        4.6.3
jupyterlab          1.2.6
notebook            6.0.3
-----
Python 3.7.6 (default, Jan  8 2020, 19:59:22) [GCC 7.3.0]
Linux-5.4.0-80-generic-x86_64-with-debian-buster-sid
8 logical CPU cores, x86_64
-----
Session information updated at 2021-08-06 11:38
```

6 Bibliografía

Documentación de Pyomo web (https://pyomo.readthedocs.io/en/stable/)

Proyecto COIN-OR del que forma parte CBC web (https://projects.coin-or.org/Cbc)

¿Cómo citar este documento?

Modelado prescriptivo para la optimización de horarios by Francisco Espiga, available under a CC BY-NC-SA 4.0 at https://www.cienciadedatos.net/documentos/py38-optimizacion-horarios-python.html

DOI 10.5281/zenodo.10006330

(https://doi.org/10.5281/zenodo.10006330)

¿Te ha gustado el artículo? Tu ayuda es importante

Mantener un sitio web tiene unos costes elevados, tu contribución me ayudará a seguir generando contenido divulgativo gratuito. ¡Muchísimas gracias! 😊