A variant of the high-school timetabling problem and a software solution for it based on integer linear programming

Iulian Ober
University of Toulouse, France
August 24, 2016
Outline

1. Problem statement
2. Problem formalization and ILP formulation
3. Tools and experimental results
4. Conclusion
Outline

1. Problem statement
2. Problem formalization and ILP formulation
3. Tools and experimental results
4. Conclusion
A particular type of High-School Timetabling problem.

Hierarchical division of student groups:

- Lectures =>
- Supervised classwork (TD) =>
- Laboratory (TP) =>

<table>
<thead>
<tr>
<th>1st year</th>
<th>2nd year</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>G1</td>
</tr>
<tr>
<td>G2</td>
<td>G2</td>
</tr>
<tr>
<td>G3</td>
<td>G3</td>
</tr>
<tr>
<td>G4</td>
<td>G4</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>
An example

<table>
<thead>
<tr>
<th>Labs</th>
<th>Classwork</th>
<th>Lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Image Description

The image shows a timetable for a course with various sessions listed for different days and times. The timetable includes sections for Labs, Classwork, and Lectures. Each section is color-coded and includes specific details such as the instructor's name, the course code, and the location. The timetable is divided into weeks, with days such as Monday, Tuesday, and Wednesday prominently displayed. The hours are listed from 8 am to 10 pm, and the days are numbered from 1 to 5. The timetable is marked as "IUT INFO - Semaine 5" and includes references to "AN" (Anesthesia), "IHM" (Introduction aux IHM), and "COD" (Gestion des projets informatiques). The timetable is updated on 19/01/2016.
Features

• predefined course-group-teacher assignment

• different room pools (per type):
  • laboratories (6), classrooms (5), multimedia rooms (2), amphitheatres (1)
  • rooms in a pool are interchangeable
  • subject to availability constraints

• dense program:
  • e.g., 33h over 4.5 days for G1A
  • potentially long days: start at 8 AM, end at 6:40 PM

• many research professors and adjunct professors (main job in industry)
  ⇒ tight availability constraints

• Potentially overlapping time slots, e.g., 8-10 AM || (8-9:30 AM, 9:30-11 AM)
  (may be modeled with disjoint time slots by subdividing them)
Features

- predefined course-group-teacher assignment

- different room pools (per type):
  - laboratories (6), classrooms (5), multimedia rooms (2), amphitheatres (1)
  - rooms in a pool are interchangeable
  - subject to availability constraints

- dense program:
  - e.g., 33h over 4.5 days for G1A
  - potentially long days: start at 8 AM, end at 6:40 PM

- many research professors and adjunct professors (main job in industry)
  ⇒ tight availability constraints

- Potentially overlapping time slots, e.g., 8-10 AM \(\parallel\) (8-9:30 AM, 9h30-11 AM)
  (may be modeled with disjoint time slots by subdividing them)
Features

- predefined course-group-teacher assignment

- different room pools (per type):
  - laboratories (6), classrooms (5), multimedia rooms (2), amphitheaters (1)
  - rooms in a pool are interchangeable
  - subject to availability constraints

- dense program:
  - e.g., 33h over 4.5 days for G1A
  - potentially long days: start at 8 AM, end at 6:40 PM

- many research professors and adjunct professors (main job in industry)
  ⇒ tight availability constraints

- Potentially overlapping time slots, e.g., 8-10 AM || (8-9:30 AM, 9h30-11 AM)
  (may be modeled with disjoint time slots by subdividing them)
Features

- predefined course-group-teacher assignment

- different room pools (per type):
  - laboratories (6), classrooms (5), multimedia rooms (2), amphitheatres (1)
  - rooms in a pool are interchangeable
  - subject to availability constraints

- dense program:
  - e.g., 33h over 4.5 days for G1A
  - potentially long days: start at 8 AM, end at 6:40 PM

- many research professors and adjunct professors (main job in industry)
  ⇒ tight availability constraints

- Potentially overlapping time slots, e.g., 8-10 AM || (8-9:30 AM, 9h30-11 AM)
  (may be modeled with disjoint time slots by subdividing them)
Motivation for this work

First of all, **practical**!

- need of convenient tooling
- keep control over the resolution engine, in order to:
  - fine-tune the solution
  - add new types of constraints as they come up

⇒ **XLSScheduler**: an automatic generator
- based on open off-the-shelf ILP solvers (Cbc, Gurobi)
- interacting via easy-to-use data format (Excel), batch style
- reasonably generic (reusable, extensible...)
Motivation for this work

First of all, practical!

- need of convenient tooling
- keep control over the resolution engine, in order to:
  - fine-tune the solution
  - add new types of constraints as they come up

⇒ **XLSScheduler**: an automatic generator
- based on open off-the-shelf ILP solvers (Cbc, Gurobi)
- interacting via easy-to-use data format (Excel), batch style
- reasonably generic (reusable, extensible...)
Outline

1. Problem statement

2. Problem formalization and ILP formulation

3. Tools and experimental results

4. Conclusion
Problem formalization

Time slots:

- **Slots** set (e.g., \{Mo0800_0930, Mo0800_1000, Mo0930_1100, ...\})
- **SlotTypes** set, \(slType : Slots \rightarrow SlotTypes\) (e.g., use duration as type)
- \(\text{rank} : Slots \rightarrow \mathbb{N}\) defines partial order between slots. E.g.:
  - \(\text{rank}(Mo1415_1545) = 4\)
  - \(\text{rank}(Mo1545_1715) = 5\)
  - \(\text{rank}(Mo1715_1845) = 6\)
  - \(\text{rank}(Mo1545_1745) = 5\)
  - \(\text{rank}(Tu0800_0930) = 8\)
- \(\text{overlap} \subseteq \text{Slot} \times \text{Slot}. \ (s_1, s_2) \in \text{overlap} \ (\text{denoted } s_1 \parallel s_2)\)
  iff slots \(s_1\) and \(s_2\) overlap chronologically. E.g.:
  - \(Mo1545_1745 \parallel Mo1545_1715\)
  - \(Mo1545_1745 \parallel Mo1715_1845\)
Problem formalization

Time slots:

- **Slots** set (e.g., \{Mo0800_0930, Mo0800_1000, Mo0930_1100, ...\})
- **SlotTypes** set, `slType : Slots \rightarrow SlotTypes` (e.g., use duration as type)
- **rank : Slots \rightarrow \mathbb{N}** defines partial order between slots. E.g.:
  - `rank(Mo1415_1545) = 4`
  - `rank(Mo1545_1715) = 5`
  - `rank(Mo1715_1845) = 6`
  - `rank(Mo1545_1745) = 5`
  - `rank(Tu0800_0930) = 8`
- **overlap \subseteq Slot \times Slot. (s_1, s_2) \in overlap** (denoted `s_1 \parallel s_2`) iff slots `s_1` and `s_2` overlap chronologically. E.g.:
  - `Mo1545_1745 \parallel Mo1545_1715`
  - `Mo1545_1745 \parallel Mo1715_1845`
Problem formalization

Time slots:

- **Slots** set (e.g., \{Mo0800_0930, Mo0800_1000, Mo0930_1100, \ldots\})
- **SlotTypes** set, \( slType : Slots \rightarrow SlotTypes \) (e.g., use duration as type)
- \( rank : Slots \rightarrow \mathbb{N} \) defines partial order between slots. E.g.:
  - \( rank(Mo1415_1545) = 4 \)
  - \( rank(Mo1545_1715) = 5 \)
  - \( rank(Mo1715_1845) = 6 \)
  - \( rank(Mo1545_1745) = 5 \)
  - \( rank(Tu0800_0930) = 8 \)
- \( overlap \subseteq Slot \times Slot. \ (s_1, s_2) \in overlap \) (denoted \( s_1 \parallel s_2 \)) iff slots \( s_1 \) and \( s_2 \) overlap chronologically. E.g.:
  - \( Mo1545_1745 \parallel Mo1545_1715 \)
  - \( Mo1545_1745 \parallel Mo1715_1845 \)
Problem formalization

Teachers:

- **Teachers** set

- \( t\text{Avail} : \text{Teachers} \times \text{Slots} \rightarrow \{0, 1\} \) defines teacher availability per time-slot.

- \( t\text{Pref} : \text{Teachers} \times \text{Slots} \rightarrow \{0, 1\} \) defines teacher preferences per time-slot.
Problem formalization

Student groups:

- *Groups* set
- *subgroup* $\subseteq *Groups* \times *Groups*. $(g_1, g_2) \in *subgroup* \text{ denoted } g_1 \prec g_2$
Problem formalization

Rooms:

- **RoomCategories set**
- \( r_{\text{Avail}} : \text{RoomCategories} \times \text{Slots} \rightarrow \mathbb{N} \) defines how many rooms of a particular category are available at each time-slot
Problem formalization

Courses:

- **Courses** set
  
  - `courseTeacher : Courses → Teachers`
  
  - `courseSlotType : Courses → SlotTypes`
  
  - `courseRoomCat : Courses → RoomCategories`
  
  - `courseRoomNb : Courses → N`
  
  - `courseGroupNb : Courses × Group → N`. `courseGroupNb(c, g)` defines how many sessions there are in the course `c` for the group `g`.
  
  - `consecCourses ⊆ Courses` (courses whose instances must be consecutive)
  
  - `precedes ⊆ Course × Course`. `(c₁, c₂) ∈ precedes` (denoted `c₁ ≪ c₂`) means all time-slots allocated to `c₁` must chronologically precede the slots of `c₂`.
  
  - `courseAvail, coursePref : Courses × Slots → {0, 1}`


Problem formalization

Courses:

- **Courses set**
- **courseTeacher : Courses → Teachers**
- **courseSlotType : Courses → SlotTypes**
- **courseRoomCat : Courses → RoomCategories**
- **courseRoomNb : Courses → N**
- **courseGroupNb : Courses × Group → N. courseGroupNb(c, g) defines how many sessions there are in the course c for the group g.**
- **consecCourses ⊆ Courses (courses whose instances must be consecutive)**
- **precedes ⊆ Course × Course. (c₁, c₂) ∈ precedes (denoted c₁ ⪯ c₂) means all time-slots allocated to c₁ must chronologically precede the slots of c₂.**
- **courseAvail, coursePref : Courses × Slots → {0, 1}**
Problem formalization

Courses:

- **Courses set**
- **courseTeacher : Courses → Teachers**
- **courseSlotType : Courses → SlotTypes**
- **courseRoomCat : Courses → RoomCategories**
- **courseRoomNb : Courses → N**
- **courseGroupNb : Courses × Group → N. courseGroupNb(c, g) defines how many sessions there are in the course c for the group g.**
- **consecCourses ⊆ Courses** (courses whose instances must be consecutive)
- **precedes ⊆ Course × Course. (c_1, c_2) ∈ precedes (denoted c_1 ≪ c_2) means all time-slots allocated to c_1 must chronologically precede the slots of c_2.**
- **courseAvail, coursePref : Courses × Slots → {0, 1}**
Problem formalization

Courses:

- **Courses set**
- courseTeacher : Courses → Teachers
- courseSlotType : Courses → SlotTypes
- courseRoomCat : Courses → RoomCategories
- courseRoomNb : Courses → N
- courseGroupNb : Courses × Group → N. courseGroupNb(c, g) defines how many sessions there are in the course c for the group g.
- consecCourses ⊆ Courses (courses whose instances must be consecutive)
- precedes ⊆ Course × Course. (c₁, c₂) ∈ precedes (denoted c₁ ≪ c₂) means all time-slots allocated to c₁ must chronologically precede the slots of c₂.
- courseAvail, coursePref : Courses × Slots → {0, 1}
Problem formalization

Courses:

- $Courses$ set
- $courseTeacher : Courses \rightarrow Teachers$
- $courseSlotType : Courses \rightarrow SlotTypes$
- $courseRoomCat : Courses \rightarrow RoomCategories$
- $courseRoomNb : Courses \rightarrow \mathbb{N}$
- $courseGroupNb : Courses \times Group \rightarrow \mathbb{N}$. $courseGroupNb(c, g)$ defines how many sessions there are in the course $c$ for the group $g$.
- $consecCourses \subseteq Courses$ (courses whose instances must be consecutive)
- $precedes \subseteq Course \times Course$. $(c_1, c_2) \in precedes$ (denoted $c_1 \ll c_2$) means all time-slots allocated to $c_1$ must chronologically precede the slots of $c_2$.
- $courseAvail, coursePref : Courses \times Slots \rightarrow \{0, 1\}$
Problem formalization

Courses:

- \textit{Courses set}
- \textit{courseTeacher} : Courses → Teachers
- \textit{courseSlotType} : Courses → SlotTypes
- \textit{courseRoomCat} : Courses → RoomCategories
- \textit{courseRoomNb} : Courses → \mathbb{N}
- \textit{courseGroupNb} : Courses × Group → \mathbb{N}. \text{courseGroupNb}(c, g) defines how many sessions there are in the course c for the group g.
- \textit{consecCourses} ⊆ Courses (courses whose instances must be consecutive)
- \textit{precedes} ⊆ Course × Course. \((c_1, c_2) ∈ \text{precedes}\) (denoted \(c_1 \ll c_2\)) means all time-slots allocated to \(c_1\) must chronologically precede the slots of \(c_2\).
- \textit{courseAvail, coursePref} : Courses × Slots → \{0, 1\}
Problem formalization

Courses:

- Courses set
- courseTeacher : Courses → Teachers
- courseSlotType : Courses → SlotTypes
- courseRoomCat : Courses → RoomCategories
- courseRoomNb : Courses → \( \mathbb{N} \)
- courseGroupNb : Courses × Group → \( \mathbb{N} \). courseGroupNb\((c, g)\) defines how many sessions there are in the course \( c \) for the group \( g \).
- consecCourses ⊆ Courses (courses whose instances must be consecutive)
  - precedes ⊆ Course × Course. \((c_1, c_2) \in \text{precedes} \) (denoted \( c_1 \ll c_2 \)) means all time-slots allocated to \( c_1 \) must chronologically precede the slots of \( c_2 \).
  - courseAvail, coursePref : Courses × Slots → \{0, 1\}
Problem formalization

Courses:

- *Courses* set
- *courseTeacher* : *Courses* → *Teachers*
- *courseSlotType* : *Courses* → *SlotTypes*
- *courseRoomCat* : *Courses* → *RoomCategories*
- *courseRoomNb* : *Courses* → ℕ
- *courseGroupNb* : *Courses* × *Group* → ℕ. *courseGroupNb(c, g)* defines how many sessions there are in the course *c* for the group *g*.
- *consecCourses* ⊆ *Courses* (courses whose instances must be consecutive)
- *precedes* ⊆ *Course* × *Course*. *(c_1, c_2) ∈ precedes* (denoted *c_1 ≪ c_2*) means all time-slots allocated to *c_1* must chronologically precede the slots of *c_2*.
- *courseAvail, coursePref* : *Courses* × *Slots* → {0, 1}
Problem formalization

Courses:

- **Courses** set
- **courseTeacher : Courses → Teachers**
- **courseSlotType : Courses → SlotTypes**
- **courseRoomCat : Courses → RoomCategories**
- **courseRoomNb : Courses → \( \mathbb{N} \)**
- **courseGroupNb : Courses × Group → \( \mathbb{N} \). courseGroupNb(c, g) defines how many sessions there are in the course c for the group g.**
- **consecCourses ⊆ Courses** (courses whose instances must be consecutive)
- **precedes ⊆ Course × Course. \((c_1, c_2) ∈ precedes\)** (denoted \(c_1 \ll c_2\)) means all time-slots allocated to \(c_1\) must chronologically precede the slots of \(c_2\).
- **courseAvail, coursePref : Courses × Slots → \{0, 1\}**
Model variables

\[ x_{s,c,g} \in \{0, 1\} \text{ for } s \in \text{Slots}, \ c \in \text{Courses}, \ g \in \text{Groups} \]

\[ x_{s,c,g} = 1 \text{ iff group } g \text{ has a course } c \text{ in time-slot } s \]
All the courses are allocated a slot:

\[ \forall c \in Courses, \forall g \in Groups : \sum_{s \in Slots} x_{s,c,g} = courseGroupNb(c, g) \]
Hard constraints

No group has two courses in parallel (nor a course in parallel with another course of one of its super-groups):

\[ \forall s \in \text{Slots}, \forall g \in \text{Groups} : \]

\[ \sum_{c \in \text{Courses}} \left( x_{s,c,g} + \sum_{g' \in \text{Groups}} x_{s,c,g'} \right) \leq 1 \]

Taking into account slot overlapping:

\[ \forall s, s' \in \text{Slots} \text{ such that } s \parallel s', \forall g \in \text{Groups} : \]

\[ \sum_{c \in \text{Courses}} \left( x_{s,c,g} + x_{s',c,g} + \sum_{g' \in \text{Groups}} (x_{s,c,g'} + x_{s',c,g'}) \right) \leq 1 \]
Hard constraints

No group has two courses in parallel (nor a course in parallel with another course of one of its super-groups):

\[ \forall s \in Slots, \forall g \in Groups : \]

\[ \sum_{c \in Courses} \left( x_{s,c,g} + \sum_{g' \in Groups \atop g < g'} x_{s,c,g'} \right) \leq 1 \]

Taking into account slot overlapping:

\[ \forall s, s' \in Slots \text{ such that } s \parallel s', \forall g \in Groups : \]

\[ \sum_{c \in Courses} \left( x_{s,c,g} + x_{s',c,g} + \sum_{g' \in Groups \atop g < g'} (x_{s,c,g'} + x_{s',c,g'}) \right) \leq 1 \]
There are enough rooms of each category for each slot:

\[ \forall s \in \text{Slots}, \forall t \in \text{RoomCategories} : \]

\[ \sum_{c \in \text{Courses}} \sum_{g \in \text{Groups}} \text{courseRoomNb}(c) \cdot x_{s,c,g} \leq r_{\text{Avail}}(t, s) \]

No precise room assignment \( \Rightarrow \) smaller model.

Extra constraint for taking into account slot overlapping
Hard constraints

There are enough rooms of each category for each slot:

\[
\forall s \in Slots, \forall t \in RoomCategories : \\
\sum_{c \in Courses \atop courseRoomCat(c) = t} \sum_{g \in Groups} courseRoomNb(c) \cdot x_{s,c,g} \leq rAvail(t, s)
\]

No precise room assignment \(\Rightarrow\) smaller model.

Extra constraint for taking into account slot overlapping
Hard constraints

There are enough rooms of each category for each slot:

\[ \forall s \in \text{Slots}, \forall t \in \text{RoomCategories} : \]

\[ \sum_{c \in \text{Courses}} \sum_{g \in \text{Groups}} \text{courseRoomNb}(c) \cdot x_{s,c,g} \leq r_{\text{Avail}}(t, s) \]

No precise room assignment \( \Rightarrow \) smaller model.

Extra constraint for taking into account slot overlapping
Hard constraints

**Consecutiveness:**

\[ \forall c \in \text{consecCourses}, \forall g \in \text{Groups}, \]
\[ \forall s, s' \in \text{Slots} \text{ such that } \text{rank}(s') - \text{rank}(s) \geq \text{courseGroupNb}(c, g) : \]
\[ x_{s,c,g} + x_{s',c,g} \leq 1 \]

**Precedence:**

\[ \forall c, c' \in \text{Courses} \text{ such that } c \ll c', \]
\[ \forall g, g' \in \text{Groups} \text{ such that } \text{courseGroupNb}(c, g) \cdot \text{courseGroupNb}(c', g') > 0, \]
\[ \forall s, s' \in \text{Slots} \text{ such that } \text{rank}(s) < \text{rank}(s') : \]
\[ x_{s,c',g'} + x_{s',c,g} \leq 1 \]
Hard constraints

Consecutiveness:

\[ \forall c \in \text{consecCourses}, \forall g \in \text{Groups}, \forall s, s' \in \text{Slots} \text{ such that } rank(s') - rank(s) \geq \text{courseGroupNb}(c, g) : \]
\[ x_{s,c,g} + x_{s',c,g} \leq 1 \]

Precedence:

\[ \forall c, c' \in \text{Courses} \text{ such that } c \ll c', \forall g, g' \in \text{Groups} \text{ such that } \text{courseGroupNb}(c, g) \cdot \text{courseGroupNb}(c', g') > 0, \forall s, s' \in \text{Slots} \text{ such that } rank(s) < rank(s') : \]
\[ x_{s,c',g'} + x_{s',c,g} \leq 1 \]
Hard constraints

Other constraints:

- The slots allocated to a Course have the right SlotType
- The teachers are available at the allocated slots
- A teacher does not have several courses in parallel
- Course availability per slot (courseAvail) is observed
Soft constraints and objective

**Soft constraint:** minimize use of unpreferred slots.

**Objective** = weighted sum of:

- cost incurred by the use of unpreferred slots of teachers

\[
UT = \sum_{\substack{s \in \text{Slots} \\ c \in \text{Courses} \\ t = \text{courseTeacher}(c) \\ g \in \text{Groups}}} (t\text{Avail}(t, s) - t\text{Pref}(t, s)) \cdot x_{s,c,g}
\]

- cost incurred by the use of unpreferred slots for courses

\[
UC = \sum_{\substack{s \in \text{Slots} \\ c \in \text{Courses} \\ g \in \text{Groups}}} (\text{courseAvail}(c, s) - \text{coursePref}(c, s)) \cdot x_{s,c,g}
\]
Soft constraints and objective

**Soft constraint:** minimize use of unpreferred slots.

**Objective** = weighted sum of:
- cost incurred by the use of unpreferred slots of teachers
  \[ UT = \sum_{s \in \text{Slots}, c \in \text{Courses}, t = \text{courseTeacher}(c), g \in \text{Groups}} (tAvail(t, s) - tPref(t, s)) \cdot x_{s,c,g} \]
- cost incurred by the use of unpreferred slots for courses
  \[ UC = \sum_{s \in \text{Slots}, c \in \text{Courses}, g \in \text{Groups}} (courseAvail(c, s) - coursePref(c, s)) \cdot x_{s,c,g} \]
Soft constraints and objective

**Soft constraint:** minimize use of unpreferred slots.

**Objective** = weighted sum of:
- cost incurred by the use of unpreferred slots of teachers
  
  \[ UT = \sum_{s \in \text{Slots}, c \in \text{Courses}, t = \text{courseTeacher}(c), g \in \text{Groups}} (tAvail(t, s) - tPref(t, s)) \cdot x_{s, c, g} \]

- cost incurred by the use of unpreferred slots for courses
  
  \[ UC = \sum_{s \in \text{Slots}, c \in \text{Courses}, g \in \text{Groups}} (courseAvail(c, s) - coursePref(c, s)) \cdot x_{s, c, g} \]
Other soft constraints: minimizing “long days”

Avoid days that start at 8AM to 6:45PM

- \( Days \) is a set of the days of week
- \( first : Days \rightarrow Slots \) maps each day to its first slot
- \( last : Days \rightarrow Slots \) maps each day to its last slot
- \( L_{d,t} \in \{0, 1\} \) for \( d \in Days \) and \( t \in Teachers \): auxiliary variable \((L_{d,t} = 1 \text{ iff } d \text{ is a “long day” for teacher } t)\)
- constraint linking \( L \) to \( x \):

\[
\forall d \in Days, t \in Teachers :

\sum_{\substack{c \in Courses \\ courseTeacher(c) = t}} \sum_{g \in Groups} (x_{first(d),c,g} + x_{last(d),c,g}) - 2 \cdot L_{d,t} \leq 1
\]

- additional cost component weighted into the cost function:

\[
LT = \sum_{t \in Teachers} \sum_{d \in Days} L_{d,t}
\]
Other soft constraints

- Minimizing “long days” for students
- Clustering busy times
- …
Outline

1. Problem statement
2. Problem formalization and ILP formulation
3. Tools and experimental results
4. Conclusion
**XLSScheduler tool**

- Batch tool based on off-the-shelf ILP solvers (Gurobi or Cbc)
- Available as online service (Cbc-based, upon request)
- Workflow:

```
Model.xlsx → XLScheduler → Timetable.xlsx
```

- Model is feasible: solution is satisfactory
- Model is infeasible: add constraints to improve quality of solution
- Solution is unsatisfactory: stop
- Solution is satisfactory: correct source of infeasibility

A web-based trial version of the tool is available at [https://www.irit.fr/~Iulian.Ober/XLScheduler/submit.php](https://www.irit.fr/~Iulian.Ober/XLScheduler/submit.php). However, due to licencing limitations, the web-based version can only use Cbc as back-end solver. This limitation severely impacts the performance of the tool, as discussed below (§4.2).
XLSScheduler tool

Input/Output format: convenient XLSX file format
2 versions of the tool:

- Python / Gurobi (Python API + solver)
  http://www.gurobi.com
- Python / PuLP / Cbc
  https://projects.coin-or.org/Cbc

- high speed (within seconds) and high quality for real-life workloads
- both solvers perform very well, but slow model creation with PuLP
Outline

1. Problem statement
2. Problem formalization and ILP formulation
3. Tools and experimental results
4. Conclusion
Conclusion

- practical approach for a relatively classical HSTT problem
- generic, extensible solution
- quick and good quality results
- possibility to troubleshoot and refine results

https://www.irit.fr/~Iulian.Ober/XLSScheduler
That’s all folks!

Thank you!

Questions?