Declarative Programming Project: Examination Timetabling

2015-2016

Presentation by Steven Adriaensen

The Exam Timetabling Problem

***Where** and **when** to schedule **which exam**, during the examination period?*

- Making a good schedule is challenging:
  - Many courses, students, lecturers and rooms.
  - Students have individual programs
  - Lecturers teach multiple courses
  - Rooms have a limited capacity/availability
  - Students and lecturers have (often conflicting) preferences (e.g. study vs. correction time)

Automating this task is an active research area.

Hard Constraints

- All exams must be scheduled exactly once.

- Exams can only take place in a **room**...
  - that **is available** for the entire period of the exam
  - whose **capacity exceeds** or equals the **number of students** attending the exam (subscribed to the course).

- **No 2 exams** can take place **at the same time**...
  - in the **same room**
  - if 1 or more **students** are **subscribed to both** courses (this includes multiple exams of the same course).
  - if the **same lecturer** teaches both courses.

Soft Constraints

- Individual preferences:
  - No exams over lunch break
  - No exams in a (multi-hour/day) period
  - Specific exam not in a (multi-hour/day) period
  - No 2 exams on the same day.
  - No 2 exams consectively.
  - Sufficient time to study for/correct all exams.

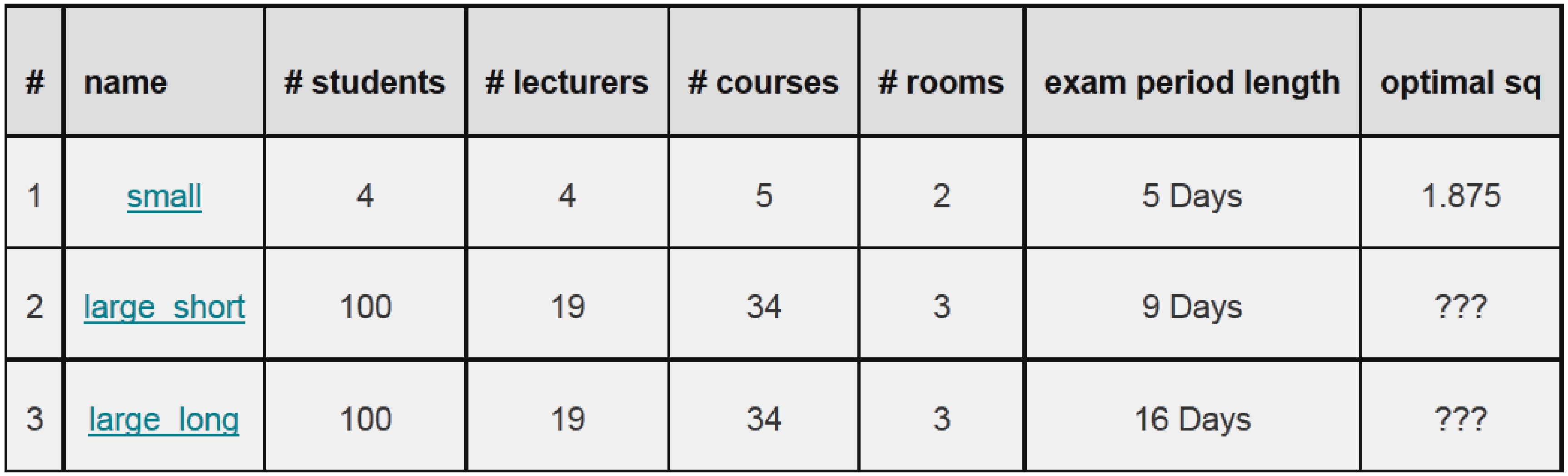
- Violating a soft-constraint ➔ penalties

- <u>Schedule cost</u>: (Normalized) Sum of penalties.

Schedule A is prefered over schedule B ⇔ cost(A) < cost(B)

Problem Instances

- We provide you with 3 problem instances, varying in size and difficulty.



#	name	# students	# lecturers	# courses	# rooms	exam period length	optimal sq
1	small	4	4	5	2	5 Days	1.875
2	large_short	100	19	34	3	9 Days	???
3	large_long	100	19	34	3	16 Days	???

- Your solver should be able to solve **any** instance.

A Concrete Problem Instance...

Given a set of courses:

```
    course(c1,'Math').
    course(c2,'Science & Technology').
    course(c3,'Philosophy').
    course(c4,'Religion').
    course(c5,'English').
```

A Concrete Problem Instance...

... having exams:

```
exam(e1,'Math').
exam(e2,'Science & Technology').
exam(e3,'Philosophy').
exam(e4,'Religion').
exam(e5,'English').

has_exam(c1,e1).
has_exam(c2,e2).
has_exam(c3,e3).
has_exam(c4,e4).
has_exam(c5,e5).
```

A Concrete Problem Instance...

Lecturers teaching courses:

```
lecturer(l1,'Mr John').
lecturer(l2,'Mr Francis').
lecturer(l3,'Mr Josef').
lecturer(l4,'Ms Ann').

teaches(l1,c1).
teaches(l1,c2).
teaches(l2,c3).
teaches(l3,c4).
teaches(l4,c5).
```

A Concrete Problem Instance...

Students following courses:

```
student(s1,'Anna').
student(s2,'Max').
student(s3,'Bill').
student(s4,'Carla').

follows(Student,c1) :- student(Student,_).
follows(Student,c2) :- student(Student,_).
follows(s2,c3).
follows(s3,c3).
follows(s1,c4).
follows(s4,c4
follows(Student,c5) :- student(Student,_).
```

A Concrete Problem Instance...

Rooms having capacities/availabilities:

```
room(r1,'Small room').
room(r2,'Large room').

capacity(r1,2).
capacity(r2,4).

availability(Room,1,10,12) :- room(Room,_).
availability(Room,2,10,12) :- room(Room,_).
availability(Room,3,10,15) :- room(Room,_).
availability(Room,4,10,12) :- room(Room,_).
availability(Room,5,10,12) :- room(Room,_).
```

A Concrete Problem Instance...

Individual preferences (i.e. soft contraints):

```
sc_lunch_break(L,1) :- lecturer(L,_).
sc_b2b(L,2) :- lecturer(L,_).
sc_no_exam_in_period(l3,3,0,24,5).
sc_no_exam_in_period(l4,Day,0,12,1) :-
   first_day(FirstDay),last_day(LastDay),
   between(FirstDay,LastDay,Day).
sc_no_exam_in_period(l1,Day,14,24,5) :-
   first_day(FirstDay),last_day(LastDay),
   between(FirstDay,LastDay,Day).
sc_not_in_period(l1,e2,1,0,24,3).
sc_correction_penalty(L,3) :- lecturer(L,_).
sc_lunch_break(S,1) :- student(S,_).
sc_same_day(S,2) :- student(S,_).
sc_b2b(S,5) :- student(S,_).
sc_study_penalty(S,3) :- student(S,_).
```

A Concrete Problem Instance...

Time required to study for/correct exams:

```
    sc_correction_time(e1,2).
    sc_correction_time(e2,1).
    sc_correction_time(e3,1).
    sc_correction_time(e4,1).
    sc_correction_time(e5,2).

    sc_study_time(e1,2).
    sc_study_time(e2,1).
    sc_study_time(e3,1).
    sc_study_time(e4,1).
    sc_study_time(e5,1).

    first_day(1).
    last_day(5).
```

A Concrete Exam Schedule

```
*** DAY 2 ***

Large room:
10:00-12:00 English (Ms Ann)

*** DAY 3 ***

Large room:
10:00-12:00 Science & Technology (Mr John)

*** DAY 4 ***

Small room:
10:00-12:00 Philosophy (Mr Francis)

Large room:
10:00-12:00 Religion (Mr Josef)

*** DAY 5 ***

Large room:
10:00-12:00 Math (Mr John)
```

A Concrete Exam Schedule

Or also:

```
schedule(
    [
    event(e4, r2, 4, 10),
    event(e1, r2, 5, 10),
    event(e3, r1, 4, 10),
    event(e5, r2, 2, 10),
    event(e2, r2, 3, 10)
    ]
)
```

Base Functionality

- Implement the following predicates:
  - `is_valid(+S)`
  - `cost(+S,?Cost)`
  - `find_optimal(-S)`
  - `find_heuristically(-S)`
  - `pretty_print(+S)`
- Use the given solution representation.
- If implemented perfectly: 18/20

Extended Functionality

- Suggested extensions:
  - `is_valid(?S)`
  - `violates_sc(+S,-SC)`
  - `is_optimal(?S)`
  - `find_heuristically(-S,+T)`
  - `pretty_print(+SID,+S)`

  `...`
- Required for 20/20, up to 3 bonus points.

Non-Functional Requirements

- Your program must work on the lab computers (E 1.4.)
- Comment your source code
- Work modular
- Find a careful balance between:
  - Declarative Style
  - Efficiency

Reporting Requirements

- Briefly explain your solution approach
- Clearly specify the strengths & weaknesses of your implementation:
  - What functionality?
  - Non-functional requirements?
- Results:
  - Optimal exam schedule for small
  - Cost of the heuristic solution for all
- Experimental results must be reproducable in under 2 min!

Deadline

- Deadline 1st term: 10th of Januari

- Deliverables:
  - Source code (+ comments)
  - Report
  - User Manual

- Project Defenses: 18, 19 and 20th of Januari