RAGIONAMENTO AUTOMATICO

Due modelli per la generazione automatica degli orari di una università durante una pandemia/epidemia

Name	Federico Bulzoni
Student No.	142242
Prof.	Agostino Dovier
Department	Informatica
Email	bulzoni.federico@spes.uniud.it
Date	27/07/2020

1. Introduzione

Il problema del *timetabling* per una università, ossia di determinare un orario per le lezioni settimanali all'interno di un'università è un problema classico nell'informatica teorica, e approcci che rendono automatica la generazione di un tale orario vengono esplorati sin dagli anni sessanta dello scorso secolo [3]. Nella sua formulazione più comune il problema richiede dato un certo numero di aule, un certo numero di professori, un certo numero di classi ed una finestra temporale, di trovare per ogni istante temporale della finestra considerata un assegnamento delle classi e dei professori nelle aule che soddisfi un insieme di vincoli. I vincoli possono essere suddivisi in *vincoli forti* e *vincoli deboli*. Considerato un possibile orario settimanale, un vincolo forte è tale per cui l'orario considerato è una soluzione valida al problema se e solo se è tale per cui tale vincolo viene soddisfatto; un esempio di vincolo forte è il fatto che non è possibile che una stessa classe svolga contemporaneamente lezione in due aule diverse. Dall'altra parte un vincolo debole è tale per cui l'orario considerato è valido anche se non soddisfa il vincolo debole, tuttavia dati due orari settimanali validi di cui uno soddisfa il vincolo debole e l'altro no, viene giudicato come *soluzione migliore* tra le due l'orario che soddisfa il vincolo debole; un esempio di vincolo debole può essere che considerando una classe, il numero di ore di lezione che svolge durante i diversi giorni della settimana dovrebbe essere equilibrato.

In questo report andremo ad analizzare una versione modificata di tale problema, nella quale si prendono in considerazione nella generazione degli orari le necessità comportate dalla convivenza con una pandemia in corso, quali ad esempio la necessità di sanificare le aule prima di ogni cambio di classe. Per tale problema vengono proposte due soluzioni distinte, una nella quale il problema viene codificato in `MiniZinc` [2] e un altra nella quale il problema viene codificato in `ASP` [1]. Le soluzioni proposte oltre a differenziarsi per il linguaggio utilizzato, si differenziano anche per l'approccio seguito, nella codifica *MiniZinc* viene fatto uso di un automa deterministico a stati finiti (**DFA**) per risolvere buona parte dei vincoli forti emersi nella modellazione del problema, mentre nella codifica *ASP* tali vincoli vengono modellati con un approccio puramente dichiarativo.

2. Specifica del problema

Si richiede di organizzare l'orario per una università di piccole dimensioni tenendo conto delle misure di contenimento in atto per una pandemia.

2.1 Dati in input

Le **aule** a disposizione sono suddivise in $G = 4$ **gruppi** distinti, questa assunzione deriva dal fatto che le aule si trovano tutte su di un corridoio con al centro delle scale, abbiamo quindi una suddivisione delle aule in base al lato del corridoio ed in base al lato in cui si trovano rispetto alle scale. Ogni gruppo, è formato dallo stesso numero di aule $K \in \mathbb{N}$, pertanto il numero totale di aule a disposizione è $G*K$. Ad ogni aula $r \in [1, G*K]$ è associata una **capacità** $cap \in [30, 60]$ che indica il numero di studenti che l'aula può contenere. Vi sono $N \in \mathbb{N}$ **coorti** di studenti da allocare. Ad ogni coorte $c \in [1, N]$ è associato il **numero di studenti** che vi sono immatricolati $nStud \in [50, 300]$, e l'**anno** di carriera $y \in [1, 3]$ corrispondente, vengono considerate solamente lauree triennali.
Le coorti sono suddivise in **dipartimenti**, sia $D \in \mathbb{N}$ il numero di dipartimenti distinti all'interno dell'università. Ad ogni coorte $c \in [1, N]$ è associato il dipartimento $dep \in [1, D]$ a cui appartiene. Infine, ad ogni coorte $c \in [1, N]$ è associato il numero di ore a settimana $reqT \in [40, 60]$ che richiede.

2.2 Requisiti

Sapendo che l'orario di apertura dell'università è dalle 8:00 alle 19:00 e lavorando con una granularità di 30 minuti, si richiede di organizzare l'orario delle lezioni rispettando i seguenti vincoli forti:

- Gli slot delle lezioni sono di 2 ore,
- ogni volta che una coorte lascia un aula c'è bisogno di una sanificazione,
- gli slot delle sanificazioni sono di 1 ora,
- per ogni istante temporale, in un gruppo di aule possono essere allocate unicamente coorti dello stesso dipartimento e sanificazioni,
- tutti gli studenti delle coorti al primo anno devono avere almeno una lezione in presenza a settimana,

- la percentuale di ore che non vengono allocate alle coorti degli anni successivi al primo rispetto a quelle richieste deve essere ben bilanciata tra i dipartimenti, in particolare si assume che le percentuali tra i diversi dipartimenti siano bilanciate con uno scarto massimo del 10%.

Si richiede di **minimizzare la non occupazione delle aule**.

2.3 Assunzioni

Si assume che i giorni di apertura siano 5: dal Lunedì al Venerdì e che alla fine di ogni giornata, immediatamente dopo la chiusura dell'università, ci sia una sanificazione generale di tutte le aule; questa assunzione implica che in un orario ottimale non ha senso avere una sanificazione all'orario di apertura (8:00) o all'orario di chiusura (19:00).
Riguardo alla richiesta di minimizzare la non occupazione delle aule, si assume che non ci siano mai istanti temporali in cui un'aula è vuota, al massimo può ritenersi *"non occupata"* quando al suo interno viene svolta una sanificazione, pertanto tale richiesta si traduce in *"minimizzare il numero di sanificazioni settimanali"*. Si assume che non porti alcun vantaggio effettuare una sanificazione in una data aula se tra l'istante prima della sanificazione e quello successivo non c'è un cambio di coorte all'interno dell'aula, pertanto questa eventualità viene impedita.

2.4 Funzione di costo

Oltre a minimizzare il numero di sanificazioni settimanali, si è scelto di dare peso anche al *numero di coorti che non riescono ad avere assegnate il numero di ore settimanali richieste* e alla *percentuale di ore mancanti rispetto alle ore settimanali richieste* per le coorti non al primo anno.
Sia K il numero di sanificazioni settimanali, Z il numero di coorti che non riescono ad avere assegnate il numero di ore richieste, e $[P_1, \ldots, P_N]$ la lista di percentuali di ore mancanti rispetto a quelle richieste per ogni coorte $i \in [1, N]$.
La funzione di costo considerata è funzione di K, Z e $[P_1, \ldots, P_N]$, in particolare si assumono le seguenti priorità: $3@Z$, $2@[P_1, \ldots, P_N]$ e $1@K$ dove con il numero intero alla sinistra della @ indichiamo la priorità associata al valore alla destra della @. Questa scelta viene giustificata, da una maggiore attenzione che si è voluta porre sul rendere l'orario più equo possibile per tutti a discapito di un possibile maggior numero di sanificazioni.

3. Scelte comuni alle due soluzioni

Considerando che l'orario di apertura dell'università è dalle 8:00 alle 19:00 e che lavoriamo con una granularità di 30 minuti, la giornata universitaria è stata suddivisa in 22 unità di tempo, dove ogni unità di tempo corrisponde a 30 minuti reali.
La soluzione al problema consiste in un **assegnamento** che ad ogni aula, ad ogni giorno considerato e ad ogni istante di tempo considerato associa un **occupante**, dove un occupante può essere una **coorte** o una **sanificazione**. Tale assegnamento viene chiamato `scheduling` in entrambe le soluzioni, il predicato $scheduling(r,d,t,o)$ se vero, indica che nell'aula $r \in [1, G*K]$, al giorno $d \in [1,5]$, all'istante $t \in [1,22]$ è presente l'occupante $o \in \{0\} \cup [1,N]$. La sanificazione viene identificata con il numero intero 0, mentre $i \in [1,N]$ indica la corrispondente coorte.
Data un aula $r \in [1, G*K]$, il gruppo $g \in [0, G-1]$ a cui appartiene è tale per cui $r\%G = g$, dove con $\%$ in questo caso indichiamo l'operazione modulo.
Riguardo al vincolo sulle coorti al primo anno, per cui si richiede che tutti gli studenti abbiano almeno una lezione in presenza a settimana, data una coorte $i \in [1,N]$ il numero di studenti che ha almeno una lezione in presenza a settimana è stato modellato come la somma delle capacità delle aule $r \in [1, G*K]$ in cui la coorte i svolge lezione, chiaramente se la coorte i viene assegnata più volte all'aula r, la capacità di r verrà considerata più volte nel calcolo.
Nei piani iniziali, l'idea era quella di introdurre come ulteriore vincolo debole il fatto che il maggior numero possibile di studenti per ogni coorte vada a lezione in presenza, tuttavia il calcolo del numero di studenti con lezioni in presenza per ogni coorte è estremamente dispendioso, soprattutto nella codifica ASP in cui assegnare il risultato di un'operazione aggregata ad un predicato, per poi effettuarci confronti sopra fa esplodere la complessità; per questi motivi l'idea è stata abbandonata in entrambe le codifiche.
Il tempo assegnato ad una coorte, così come quello assegnato alle operazioni di sanificazione, non è altro che il conteggio delle volte in cui la coorte (o sanificazione) viene assegnata ad una tripla $(r,d,t) \in [1, G*K] \times [1,5] \times [1,22]$, dove r è un'aula, d è un giorno della settimana e t è un istante temporale.
Il vincolo:

Vincolo aule-dipartimenti

Per ogni istante temporale, in un gruppo di aule possono essere allocate unicamente coorti dello stesso dipartimento e sanificazioni.

È stato modellato in entrambe le soluzioni tramite il predicato al primo ordine:

$$\forall r \forall dep \forall t (scheduling(r,d,t,c) \wedge department(c,dep) \rightarrow \\ \neg \exists r' (r' \neq r \wedge scheduling(r',d,t,c') \wedge r\%G = r'\%G \wedge department(c',dep') \wedge dep' \neq dep)) \tag{3.1}$$

dove $r, r' \in [1, G*K]$, $d \in [1,5]$, $t \in [1,22]$, $c, c' \in [1,N]$, $dep, dep' \in [1,D]$.

Il vincolo:

Vincolo coorti al primo anno - studenti

Tutti gli studenti delle coorti al primo anno devono avere almeno una lezione in presenza a settimana.

È stato modellato in entrambe le soluzioni tramite il predicato al primo ordine:

$$\forall c(year(c,1) \wedge satisfiedStudents(c,x) \wedge nStudents(c,y) \rightarrow x \geq y) \tag{3.2}$$

dove $c \in [1,N]$, $year(c,1)$ se e solo se la coorte c è al primo anno, $satisfiedStudents(c,x)$ è vero se e solo se il numero di studenti che hanno lezione in presenza della coorte c è uguale ad x ed x viene calcolato come specificato in precedenza, e $nStudents(c,y)$ è vero se e solo se il numero di studenti della coorte c è pari ad y.

Il vincolo:

Vincolo bilanciamento insoddisfazione dipartimenti

La percentuale di ore che non vengono allocate alle coorti degli anni successivi al primo rispetto a quelle richieste deve essere ben bilanciata tra i dipartimenti, in particolare si assume che le percentuali tra i diversi dipartimenti siano bilanciate con al più uno scarto del 10%.

è quello che ha creato più problemi nella fase di modellazione e anche di implementazione dati i calcoli non basilari che richiede. In particolare richiede di:

1. isolare i dipartimenti per cui esiste almeno una coorte non al primo anno di studi che vi appartiene,
2. calcolare il numero totale di unità di tempo richieste per ogni dipartimento considerando solamente le coorti non al primo anno,
3. calcolare il numero totale di unità di tempo assegnate per ogni dipartimento considerando solamente le coorti non al primo anno,
4. calcolare la percentuale di ore assegnate alle coorti al primo anno rispetto a quelle richieste (o equivalentemente non assegnate) per ognuno dei dipartimenti,
5. verificare che non esistano due dipartimenti diversi per cui il valore assoluto della differenza tra le loro due percentuali così calcolate superi il 10%.

Per questo vincolo non staremo a dare la formalizzazione in logica al primo ordine, ma è facilmente ricavabile dalla procedura appena descritta.

Per entrambe le soluzioni si è cercato di utilizzare il minor numero di vincoli possibile cercando al contempo di tenere intatta la semantica delle soluzioni, per questo motivo tutti gli altri vincoli che possono essere raggruppati come *vincoli riguardanti la regolarità* dello scheduling, sono stati condensati in entrambe le soluzioni in un unico vincolo che può essere chiamato **vincolo di regolarità**. Tale vincolo può essere descritto come:

Vincolo di regolarità

Non può essere che una sanificazione sia schedulata al primo o all'ultimo istante di una giornata, dato che a fine giornata avviene la sanificazione generale di tutte le aule. Una sanificazione dura 2 unità di tempo (1 ora) e viene effettuata **se e solo se** tra l'istante prima dell'inizio della sanificazione e l'istante successivo alla sanificazione all'interno della stanza interessata c'è stato un cambio di coorte. Uno slot di lezione dura 4 unità di tempo (2 ore).

La soluzione in `MiniZinc` e quella in `ASP` si differenziano principalmente per come il vincolo di regolarità è stato implementato.

4. La soluzione in MiniZinc

La soluzione in `MiniZinc` non ha creato grossi problemi durante la fase implementativa, anche se è passata attraverso diversi refactoring. La prima versione dell'implementazione in MiniZinc lavorava con una ricerca binaria, e il predicato *scheduling* era implementato come un'array di variabili booleane associate alle tuple (r,d,t,o) dove r è un'aula, d è un giorno, t è un istante temporale e o è una coorte o una sanificazione. Tale versione tuttavia si è rivelata generalmente meno performante della successiva versione che lavora con una ricerca intera ed in cui il predicato *scheduling* è implementato come un array di occupanti (sanificazione o una coorte), questo risultato non è stato una sorpresa dato che passare ad una versione di scheduling fatta in questo modo ha permesso di utilizzare constraint globali implementati nella libreria `globals.mzn` quali ad esempio `count` e `regular` in modo più semplice e si ipotizza efficiente. Un altro refactoring importante attraversato da questa soluzione ha riguardato il DFA utilizzato per implementare il vincolo di regolarità, si rimanda alla prossima sezione per i dettagli.

4.1 Implementazione del vincolo di regolarità

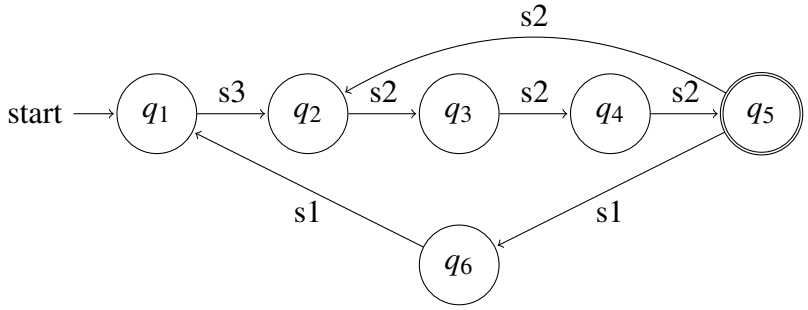

Figure 4.1: DFA alla base della soluzione MiniZinc.

Il vincolo di regolarità è stato implementato in modo naturale sfruttando il constraint globale `regular` offerto dalla libreria `globals.mzn`. Il constraint globale `regular` consente di definire un automa deterministico a stati finiti (DFA) per determinare se una data parola appartiene o meno al linguaggio definito dell'automa, nel nostro caso per ogni giorno della settimana ed ogni aula, deve essere verificato che lo scheduling sia valido rispetto al vincolo di regolarità definito nel capitolo 3.
L'automa implementato è derivato da quello mostrato in figura 4.1. L'automa presentato in figura 4.1 lavora sull'alfabeto $\Sigma = \{s1, s2, s3\}$ dove con $s1$ viene indicato che l'occupante che si sta leggendo è una sanificazione, con $s2$ viene indicato che l'occupante che si sta leggendo è una coorte e coincide con la precedente coorte nell'aula, con $s3$ viene indicato che l'occupante che si sta leggendo è una coorte ed è una coorte diversa da quella che stava occupando in precedenza

l'aula. Si noti come tale automa forzi lo scheduling per una data aula in un dato giorno a seguire il vincolo di regolarità: se ci si trova sullo stato q_1 non si può leggere altro se non una coorte diversa da quella vista in precedenza, per poi rivedere tale corte altre tre unità temporali di fila fino ad arrivare allo stato finale q_5, in tale modo viene codificato il fatto che uno slot di lezione dura 4 unità di tempo (2 ore). Successivamente dallo stato finale q_5 o si rilegge ancora la stessa coorte vista in precedenza tornando allo stato q_2 per poi tornare allo stato q_5 dopo aver visto altre tre occorrenze di tale coorte, oppure si legge una sanificazione che porta allo stato q_6 da cui si richiede di leggere un'altra sanificazione per tornare allo stato q_1, in questo modo viene codificato il fatto che una sanificazione prende 2 unità di tempo (1 ora), inoltre si noti che da q_1 si richiede di leggere una coorte diversa da quella vista in precedenza, grazie a questa richiesta viene garantito che una sanificazione possa essere schedulata solamente quando c'è un cambio effettivo di coorte all'interno dell'aula. Il fatto che non può venire schedulata una sanificazione all'inizio ed alla fine di una giornata è codificato dal fatto che l'unico stato finale sia q_5 al quale si può arrivare solamente dopo aver riconosciuto uno slot di lezione assegnato ad una coorte.

In una prima versione dell'implementazione veniva effettivamente calcolata una matrice `shift` di valori in Σ basandosi sui valori di `scheduling`, tuttavia tale versione lavorava su una versione semplificata del vincolo di regolarità in cui non veniva forzato il fatto che una sanificazione può avvenire **se e solo se** c'è un cambio di coorte all'interno dell'aula.
Nella successiva versione dell'implementazione è stato scelto di abbandonare la matrice `shift`, eliminando così una matrice di $(G*K)*(5*22)$ variabili, in favore di una versione aumentata dell'automa che fosse in grado di calcolare al volo il corrispettivo valore in Σ, il compromesso per fare ciò è stato un aumento lineare rispetto al numero di coorti del numero di stati dell'automa. Questa versione dell'automa ha permesso anche di implementare la versione completa del vincolo di regolarità di cui abbiamo parlato in precedenza.
L'idea è stata quella di lavorare su un automa i cui stati sono il prodotto tra i sei stati presentati nell'automa in figura 4.1, e i possibili occupanti, uno stato dell'automa definitivo è dunque una coppia (q,o) dove $q \in [1,6]$ è il corrispondente stato del DFA in figura 4.1 e $o \in [0,N]$ è un occupante dove come al solito con 0 viene indicata una sanificazione e con $0 < i \leq N$ viene indicata la corrispettiva coorte. Il passaggio a questa versione dell'implementazione ha permesso un risparmio in termini di tempo di esecuzione per trovare la prima soluzione al problema ed un netto miglioramento della qualità delle soluzioni trovate.

4.2 La funzione di costo

La funzione di costo descritta in 2.4 è stata implementata come una somma pesata nella quale i tre valori interessati vengono tutti normalizzati in modo da ricadere all'interno dell'intervallo $[0.0, 1.0]$, ognuno dei valori normalizzati viene moltiplicato per un coefficiente pari alla priorità riportata in 2.4. Riguardo alle percentuali di ore insoddisfatte tra le coorti viene considerato il valore massimo di tale percentuale tra tutte le coorti.

4.3 Annotazioni per la ricerca

Come riportato in precedenza dopo aver provato ad utilizzare una ricerca binaria, si è optato per utilizzare una ricerca intera. Tale ricerca nel tentativo di ottimizzarla è stata annotata, viene

riportata nel codice l'annotazione che ha ottenuto i risultati migliori.

4.4 Il codice

Viene qui riportato il codice della soluzione in MiniZinc, le parti relative alla generazione dell'output sono omesse.

```
% post-pandemic_timetable.mzn
include "globals.mzn";

% Parameters:
int : G = 4; % Number of different groups of rooms.
int : K; % Number of rooms in each group of rooms.
set of int : ROOM = 1 .. G*K;
array[ROOM] of int : capacity; % Association between a room and its
    capacity.

int : N; % Number of different coortes
set of int : COORTE = 1 .. N;
array[COORTE] of 50 .. 300 : nStudents; % Association between a coorte
    and its number of students.
array[COORTE] of 1 .. 3 : year; % Association between a coorte and the
    associated year of study.

int : D; % Number of different departments.
set of int : DEPARTMENT = 1 .. D;
array[COORTE] of DEPARTMENT : department; % Association between a coorte
     and the department to which it belongs.

int : TIMEUNIT_X_DAY = 22; % Number of timeunit in each day, a timeunit
    corresponds to 0.5 hours.
set of int : TIMEUNIT = 1 .. TIMEUNIT_X_DAY;
int : N_DAYS = 5; % Number of days considered for the scheduling.
set of int : DAY = 1 .. N_DAYS;

array[COORTE] of 40 .. 60  : requiredTime; % Association between a
    coorte and the minimum amount of lecture's hours that it requires.

% Decision variables:
set of int : OCCUPANT = 0 .. N; % 0 is the sanification.
% A table where for each room, day and timeunit it associates the
    occupant of the room in that day and timeunit. The room can be
    occuped by a coorte or by a sanification.
array[ROOM, DAY, TIMEUNIT] of var OCCUPANT : scheduling;

% Utilities:
predicate sanification(var OCCUPANT : occupant) =
  occupant == 0;

predicate sameGroup(ROOM : room1, ROOM : room2) =
  room1 mod G == room2 mod G;

predicate sameDep(var COORTE : coorte1, var COORTE : coorte2) =
  department[coorte1] == department[coorte2];
```

```
40
41 function var int : timeAssigned(OCCUPANT : occupant) =
42    count([scheduling[room, day, timeunit] | room in ROOM, day in DAY,
      timeunit in TIMEUNIT], occupant);
43
44
45 function var int : satisfiedStudents(COORTE : coorte) =
46   sum(room in ROOM)( (count([scheduling[room, day, timeunit] | day in
      DAY, timeunit in TIMEUNIT], coorte) div 4) * capacity[room] );
47
48 % Constraints:
49 % Automata declaration.
50 int : Q = 6 * (N + 1);
51
52 % Function that given a pseudo-state s and an occupant o gives the
     actual state of the automata.
53 function int : buildState (int : s, int : o) =
54   (s - 1)*(N + 1) + (o + 1);
55
56 int : q0 = buildState(1,0);
57 set of int : STATE = 1 .. Q;
58 int : S = N+1;
59
60 % Transition function, the following properties are required:
61   % 1. A lesson's slot is of 4 time units.
62   % 2. When a coorte left a classroom, a sanification is needed. A
      sanification's slot is of 2 time units.
63   % 3. Optimization properties:
64     % a. A sanification must be done if and only if there is a change of
      coorte in the considered classroom.
65     % b. Sanification can't be done at the start or at the end of the
     day, since we assume that after the end of the last lesson of the day
      a sanification is made by default in eache room.
66 array[STATE, OCCUPANT] of int : t = array2d(STATE, OCCUPANT,
67   [
68     if state = 1
69       then
70       if oldOcc = 0 \/ (occ != 0 /\ oldOcc != occ)
71         then buildState(2, occ)
72         else 0
73       endif
74     elseif state > 1 /\ state < 5
75       then
76       if occ != 0 /\ oldOcc = occ
77         then buildState(state+1, oldOcc)
78         else 0
79       endif
80     elseif state = 5
81       then
82       if occ = 0
83         then buildState(6, oldOcc)
84       elseif occ != 0 /\ occ = oldOcc
85           then buildState(2, oldOcc)
86       else 0
87       endif
88     else % stato 6.
```

```
      if occ = 0
        then buildState(1, oldOcc)
        else 0
      endif
    endif
  | state in 1..6, oldOcc in OCCUPANT, occ in OCCUPANT]);

set of int : FINAL = {buildState(5, o) | o in OCCUPANT};

% Regularity constraint:
constraint
  forall (room in ROOM, day in DAY)(
     regular([scheduling[room, day, timeunit] | timeunit in TIMEUNIT], Q
    , OCCUPANT, t, q0, FINAL)
  );

% Coortes of the same department must be allocated in the same group of
    rooms.
constraint
  forall(room in ROOM, day in DAY, timeunit in TIMEUNIT, dep in
    DEPARTMENT)(
    let
    {var OCCUPANT : occupant = scheduling[room, day, timeunit]} in
    not sanification(occupant) /\ department[occupant] = dep
    ->
    not exists(room2 in ROOM)
    (
      let
      {var OCCUPANT : occupant2 = scheduling[room2, day, timeunit]}
      in not sanification(occupant2) /\ sameGroup(room, room2) /\ not
    sameDep(occupant, occupant2)
    )
  );


% All the students at their first year must have a lecture.
constraint
  forall(coorte in COORTE)(
    year[coorte] = 1 -> satisfiedStudents(coorte) >= nStudents[coorte]
  );

% Restriction to the coortes at the second and third year, and to the
    departments with at least one of these coortes.
set of DEPARTMENT : DEPWITHNEXTYEARS = {dep | dep in DEPARTMENT where
    exists(coorte in COORTE)(department[coorte] = dep /\ year[coorte] !=
    1)};
set of COORTE : COORTENEXTYEAR = {coorte | coorte in COORTE where year[
    coorte] > 1};

function int : totRequiredTimeDep(DEPWITHNEXTYEARS : dep) =
  sum(coorte in COORTENEXTYEAR where department[coorte] = dep)(
    requiredTime[coorte]
  );

function var int : timeAssignedDep(DEPWITHNEXTYEARS: dep) =
  sum(coorte in COORTENEXTYEAR where department[coorte] = dep)(
```

```
137     timeAssigned(coorte)
138   );
139
140 function var float : percSatDep(DEPWITHNEXTYEARS : dep) =
141   timeAssignedDep(dep) * 100 / totRequiredTimeDep(dep);
142
143 % Constraint on the percentage of unsatisfied hours between departments.
144 constraint
145   forall(dep1, dep2 in DEPWITHNEXTYEARS where dep1 != dep2 /\
      timeAssignedDep(dep1) < requiredTime[dep1] /\ timeAssigned(dep2) <
      requiredTime[dep2])(
146     abs(percSatDep(dep1) - percSatDep(dep2)) < 10
147   );
148
149 % SEARCH:
150 % Objective function.
151 % Priorita' 1: tempo di sanificazione (timeAssigned(0)).
152 % Priorita' 2: percentuale insoddisfazione.
153 % Priorita' 3: numero di coorti insoddisfatte.
154 function var float : sanificationTime() =
155   timeAssigned(0);
156
157 function var int : nCoortiUnsat() =
158   length([coorte | coorte in COORTE where timeAssigned(coorte) <
      requiredTime[coorte]]);
159
160 function var float : percUnsat(COORTE : coorte) =
161   if timeAssigned(coorte) >= requiredTime[coorte]
162     then 0
163     else (requiredTime[coorte] - timeAssigned(coorte)) / requiredTime[
      coorte]
164   endif;
165
166 solve :: int_search(scheduling, dom_w_deg, indomain_random)  minimize (
      timeAssigned(0)/(TIMEUNIT_X_DAY * N_DAYS * G * K) + 3*(nCoortiUnsat()
      /N) + 2*max([percUnsat(coorte) | coorte in COORTE]));
```

5. La soluzione in ASP

La soluzione in ASP è quella che più ha dato problemi nella fase implementativa, i problemi principali sono sorti nell'utilizzo degli aggregati, in particolare nell'utilizzo di `#sum`. Andando a vedere la documentazione di `clingo` [1], la causa di tali problemi è stata individuata e consiste nel modo in cui avviene la valutazione di predicati nel quale il risultato di un aggregato viene prima assegnato ad una variabile (e/o ad un predicato) e poi confrontato con operatori relazionali. Si consideri il seguente codice ASP:

```
{a; b; c}.
:- #sum{1 : a; 2 : b; 3 : c} = N, N > 3.
```

questo codice durante la valutazione dà luogo a tre vincoli di integrità:

```
#sum{1 : a; 2 : b; 3 : c} = 4.
#sum{1 : a; 2 : b; 3 : c} = 5.
#sum{1 : a; 2 : b; 3 : c} = 6.
```

tale comportamento può essere evitato confrontando direttamente il valore di `#sum` invece di assegnarlo prima alla variabile `N`, tuttavia non sempre ciò è fattibile.
Si consideri il vincolo sul bilanciamento delle percentuali di ore insoddisfatte tra i dipartimenti, tale vincolo richiede necessariamente di calcolare ed assegnare a delle variabili il tempo totale di lezione in presenza richiesto da ogni dipartimento ed il tempo totale che gli viene effettivamente assegnato; tali valori vengono confrontati tra loro in quanto al fine del vincolo in questione vengono considerati unicamente i dipartimenti per cui il tempo di lezione in presenza che gli viene assegnato è minore del tempo da loro richiesto, infine presi due dipartimenti diversi deve anche essere calcolata la differenza tra le percentuali di soddisfazione (eq. insoddisfazione) delle ore richieste tra i due dipartimenti e tale differenza deve essere verificata essere minore o uguale al 10%.
Per le stesse motivazioni è stato evitato di considerare nella funzione di costo la percentuale di studenti che non riesce ad avere lezioni in presenza, in quanto tale calcolo farebbe esplodere il tempo di esecuzione del modello.

5.1 Implementazione del vincolo di regolarità

Il vincolo di regolarità è stato implementato attraverso una definizione esaustiva e mutuale di ciò che consideriamo come slot di lezione valido e ciò che consideriamo come slot di sanificazione valido. Le definizioni di `lectureSlot/4` e `sanificationSlot/3` derivano direttamente dalla definizione del vincolo di regolarità descritto nel capitolo 3. Una volta definiti `lectureSlot/4` e `sanificationSlot/3` l'unico vincolo inserito nel modello a riguardo è che si richiede che al primo istante di ogni giornata, in ogni aula, sia pianificato uno slot di lezione; data la natura

mutuale che caratterizza la definizione di `lectureSlot/4` e `sanificationSlot/3` segue la corretta implementazione del vincolo di regolarità.

5.2 La funzione di costo

La funzione di costo descritta in 2.4 è stata implementata utilizzando la sintassi offerta dal linguaggio per specificare i vincoli deboli, le priorità dei tre valori interessati sono specificate direttamente all'interno della specifica dei singoli vincoli deboli. In questo caso viene considerato l'ordine lessicografico delle tuple $(Z,S,K) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ dove Z è il numero di coorti a cui non è assegnato almeno il numero di ore da loro richieste, S è la somma delle percentuali di ore di lezione in presenza mancanti rispetto a quelle richieste tra i vari dipartimenti e K è il numero di unità di tempo assegnate alle sanificazioni.

5.3 Il codice

Viene qui riportato il codice della soluzione in ASP, le parti relative alla generazione dell'output sono omesse.

```
#const g = 4.
% k is in input
room(1..k*g).
% capacity(r, c | room(r)) in input
% n in input
coorte(1..n).
% nStudents(c, n | coorte(c)) in input
% year(c, y | coorte(c)) in input
% d in input
dep(1..d).
% department(c, d | coorte(c), dep(d)) in input
#const timeunit_x_day = 22.
timeunit(1 .. timeunit_x_day).
#const n_days = 5.
day(1..n_days).
% requiredTime(c, t | coorte(c)) in input.

occupant(0..n).

% Specifiche per limitare i domini.
#const min_cap = 30.
#const max_cap = 60.
cap(min_cap .. max_cap).

#const min_stud = 50.
#const max_stud = 300.
stud(min_stud .. max_stud).

y(1..3).
day(1..n_days).
#const min_reqT = 40.
#const max_reqT = 60.
reqT(min_reqT .. max_reqT).
```

```
34
35 tAss(0..timeunit_x_day*n_days*g*k).
36
37 % esiste uno e solo un occupant associato ad ogni R,D,T.
38 1 {scheduling(R,D,T,O) : occupant(O)} 1 :- room(R), day(D), timeunit(T).
39
40 sameGroup(R1, R2) :- R1 \ g = R2 \ g,
41             room(R1), room(R2).
42 sameDep(C1, C2) :- department(C1,D1), department(C2,D2),
43                    D1 = D2,
44                    coorte(C1), coorte(C2).
45
46 sanificationTime(K) :- K = #count{R,D,T : scheduling(R,D,T,0), room(R),
       day(D), timeunit(T)}.
47
48 % in un gruppo di stanze ad un certo istante temporale devono essere
       presenti solamente coorti dello stesso dipartimento.
49 :- day(D), timeunit(T),
50    scheduling(R1, D, T, C1), scheduling(R2, D, T, C2),
51    C1 != C2,
52    not sameDep(C1, C2), sameGroup(R1, R2),
53    room(R1), room(R2), coorte(C1), coorte(C2).
54
55 % definizione di lectureSlot e sanificationSlot.
56 lecPreviousBad(R, D, T, C) :- T != 1, not sanificationSlot(R, D, T-2),
       not lectureSlot(R,D,T-4,C),
57                                room(R), day(D), timeunit(T), coorte(C).
58
59 lecFollowingBad(R, D, T, C) :- T + 4 <= timeunit_x_day, not
       sanificationSlot(R, D, T+4), not lectureSlot(R, D, T+4, C),
60                                 room(R), day(D), timeunit(T), coorte(C).
61
62 lectureSlot(R, D, T, C) :- scheduling(R, D, T, C), scheduling(R, D, T +
       1, C), scheduling(R, D, T + 2, C), scheduling(R, D, T + 3, C),
63                             not lecPreviousBad(R, D, T, C),
64                             not lecFollowingBad(R, D, T, C),
65                             room(R), day(D), timeunit(T), coorte(C).
66
67 sanificationSlot(R, D, T) :- scheduling(R, D, T, 0), scheduling(R, D, T
       + 1, 0),
68                               lectureSlot(R, D, T - 4, C1),
69                               lectureSlot(R, D, T + 2, C2),
70                               C1 != C2,
71                               coorte(C1), coorte(C2),
72                               room(R), day(D), timeunit(T).
73
74 % vincolo di regolarita', basta questo.
75 :- room(R), day(D), not lectureSlot(R,D,1,C) : coorte(C).
76
77 % vincolo sulle coorti al primo anno.
78 :- coorte(C), year(C,1), nStudents(C,Y),
79    #sum{Z, R,D,T : lectureSlot(R,D,T,C), capacity(R,Z), cap(Z), room(R),
        day(D), timeunit(T)} < Y.
80
81 timeAssigned(C, X) :- X = #count{R,D,T : scheduling(R,D,T,C), room(R),
       day(D), timeunit(T)},
```

```
                        coorte(C).

unsatisfied(C) :- coorte(C), requiredTime(C,Y), reqT(Y), #count{R,D,T :
    scheduling(R,D,T,C), room(R), day(D), timeunit(T)} < Y.

nUnsat(Z) :- Z = #count{C : coorte(C), unsatisfied(C)}.

% preparazione al vincolo di bilanciamento tra dipartimenti.
depWithNextYears(D) :- 1 {coorteNextYear(C) : department(C,D)} , dep(D).
coorteNextYear(C) :- coorte(C), year(C,X), y(X), X > 1.

totRequiredTimeDep(DI, Z) :- Z = #sum{Y, C : coorteNextYear(C),
    requiredTime(C,Y), reqT(Y), department(C, DI)},
                            depWithNextYears(DI).

timeAssignedDep(DI, Z) :- Z = #count{C, R,D,T : coorteNextYear(C),
    department(C,DI), scheduling(R,D,T,C), room(R), day(D), timeunit(T)},
                        depWithNextYears(DI).

% vincolo di bilanciamento tra i dipartimenti.
:- depWithNextYears(DI1), depWithNextYears(DI2), DI1 != DI2,
    totRequiredTimeDep(DI1, X1), totRequiredTimeDep(DI2, X2),
    timeAssignedDep(DI1, Y1), timeAssignedDep(DI2, Y2),
    Y1 < X1, Y2 < X2,
    | (Y1*100 / X1) - (Y2*100 / X2) | > 10.


% funzione di costo.
:~ nUnsat(Z). [Z@3]
:~ coorte(C), timeAssigned(C,X), tAss(X), requiredTime(C,Y), reqT(Y),
    X < Y,
    RES = (Y-X)*100/Y. [RES@2]
:~ sanificationTime(K). [K@1]
```

6. Esperimenti

Per testare i due modelli è stata progettata una suite di tests generati prevalentemente in modo casuale. Sono state generate 100 istanze totali su cui entrambi i modelli sono stati lanciati per 5 minuti sfruttando 4 thread su ogni istanza e da cui sono stati ricavati i 3 valori coinvolti per il calcolo della funzione di costo descritta in 2.4.
Delle 100 istanze generate, per 50 di esse K, il numero di aule per ognuno dei 4 gruppi di aule è stato fissato a 1 (per un totale di 4 aule), e per le rimanenti 50 istanze è stato fissato a 2 (per un totale di 8 aule). La scelta di limitare K ad un valore massimo pari a 2 è derivato dai test svolti durante l'implementazione dei due modelli; in particolare a limitare il valore di K sono stati i risultati forniti dal modello ASP che per valori più grandi di K non ha ottenuto risultati soddisfacenti.
Il valore di N, il numero di coorti, è generato casualmente tra $G * K$ (il numero di aule) e 16, in questo caso è stato il modello MiniZinc a fornire l'indicazione per il valore massimo di N, in quanto con valori più alti di 16 il modello incominciava ad ottenere risultati insoddisfacenti. Il numero di dipartimenti diversi, D, è stato fissato a 3 per tutte le istanze, questa scelta è dettata dal fatto che il vincolo di bilanciamento tra le percentuali di insoddisfazione dei diversi dipartimenti è quello più oneroso dal punto di vista computazionale, soprattutto per l'implementazione ASP. Gli altri parametri, quali le capacità delle aule, il numero di studenti delle coorti, l'anno delle coorti, il dipartimento delle coorti, ed il tempo richiesto da ogni coorte sono generati casualmente all'interno del range specificato nelle codifiche ASP e MiniZinc.

Nelle tabelle 8.1, 8.2, 8.3, 8.4 sono riportati i risultati dell'esecuzione dei due modelli sulle 100 istanze generate. I valori riportati sono i parametri K ed N, nUS che riporta il numero di coorti alle quali non sono state assegnate le ore di lezione in presenza da loro richieste, $avg(US)$ (resp. $max(US)$) ossia il valore medio della percentuale di insoddisfazione tra i dipartimenti (resp. il valore massimo) e nUS che indica il numero di unità di tempo assegnate alle sanificazioni. Le celle vuote indicano che il rispettivo modello non ha trovato nessuna soluzione valida.

6.1 Commenti sui risultati degli esperimenti

Dai risultati ottenuti emerge che il modello ASP è più efficace nel trovare una soluzione valida al problema di quanto non lo sia il modello MiniZinc. Tuttavia, durante la fase implementativa era emerso che il modello MiniZinc lavora meglio su valori di K più alti e peggio su valori di N più alti, un ipotesi per spiegare questo comportamento riguarda l'automa utilizzato per riconoscere il vincolo di regolarità, che ricordiamo ha un numero di stati che aumenta linearmente all'aumentare di N. Dall'altra parte invece, il modello ASP soffre maggiormente all'aumentare di K e conseguentemente del numero di aule, dato che un grosso collo di bottiglia di tale implementazione pare essere il vincolo riguardante le coorti al primo anno i cui studenti devono avere

almeno una lezione in presenza a settimana.
Va anche detto che considerare solamente K ed N come parametri porta ad una grossa perdita di informazione, dato che non stiamo considerando quanti studenti ha ogni coorte, i dipartimenti a cui sono assegnate, il numero di ore che richiedono e quante sono al primo anno.
Un'altra osservazione riguarda la qualità dei risultati, che sembrerebbe essere più alta per istanze semplici nei risultati ottenuti dal modello ASP, per poi migliorare nei risultati ottenuti dal modello MiniZinc all'aumentare della complessità delle istanze (anche se il modello MiniZinc trova una soluzione con maggiore difficoltà rispetto al modello ASP).

7. Considerazioni e conclusioni

In questo report abbiamo mostrato due modelli per risolvere un problema di grande attualità quale è quello della generazione degli orari universitari considerando le necessità comportate da una pandemia in corso, i due modelli hanno prodotto risultati soddisfacenti riuscendo a generare effettivamente degli orari per le lezioni validi rispetto alle nostre specifiche sulla maggior parte delle istanze considerate.
Purtroppo non è stato possibile raggiungere l'ottimo globale per nessuna delle istanze considerate, e dunque non abbiamo dati empirici per provare ad inferire la complessità del problema trattato, tuttavia è risaputo che il problema sottostante al problema considerato è NP-completo in quasi ogni sua variante. [3]
Certamente i due modelli non sono privi di problemi, primo su tutti il fatto che non vengono considerate per le coorti non al primo anno quanti studenti non riescono ad avere lezioni in presenza. I modelli potrebbero essere resi più efficienti rilassando qualche vincolo, ad esempio il vincolo di regolarità, e ancora meglio, potrebbero essere resi più efficienti assegnando di default qualche coorte a delle aule predefinite, per esempio all'inizio del primo giorno della settimana, in modo da tagliare lo spazio di ricerca considerato.

8. Tabelle dei risultati

Table 8.1: Risultati dell'esecuzione dei due modelli sulle istanze di test da 1 a 25.

Parametri		ASP			MiniZinc		
K	N	nUS	avg(US)	nSan	nUS	max(US)	nSan
1	4	0	0	40	0	0	40
1	4	0	0	40	0	0	48
1	4	0	0	40	0	0	44
1	4	0	0	40	0	0	44
1	5	0	0	40	0	0	52
1	5	0	0	40	0	0	44
1	5	0	0	40	0	0	48
1	5	0	0	40	0	0	44
1	6	0	0	40	0	0	48
1	6	0	0	40	0	0	40
1	6	0	0	40	0	0	44
1	6	0	0	40	0	0	44
1	6	0	0	40	1	2	44
1	7	1	21	52	1	2	40
1	7	2	17	80	5	13	40
1	7	0	0	52	0	0	44
1	7	0	0	68	5	14	40
1	7	0	0	64	1	2	40
1	8	3	26	52	5	27	40
1	8	4	44	64	6	6	40
1	8	3	58	60	6	8	40
1	8	2	43	48	6	20	40
1	8	3	31	48	5	4	40
1	8	2	38	76	5	14	40
1	9	5	53	64	8	28	40

Table 8.2: Risultati dell'esecuzione dei due modelli sulle istanze di test da 26 a 50.

Parametri		ASP			MiniZinc		
K	N	nUS	avg(US)	nSan	nUS	max(US)	nSan
1	9	5	55	52	7	30	40
1	9	4	58	60	8	17	40
1	9	4	73	72	9	22	40
1	10	5	80	60	7	42	40
1	10	7	42	40	10	33	40
1	10	5	43	44	7	40	40
1	10	5	57	40	9	37	40
1	10	6	59	52	10	32	40
1	12	9	59	64	12	53	40
1	12	9	56	56	11	47	40
1	12	7	63	60	10	50	40
1	12	9	60	56	11	50	40
1	13	9	32	40	12	54	40
1	13	10	33	68	12	64	40
1	13	10	33	60	12	52	40
1	13	9	42	40	11	55	40
1	13	9	59	56	12	59	40
1	14	11	38	40	12	66	40
1	14	11	26	44	12	70	40
1	15	12	52	96	15	64	40
1	15	12	36	52	15	68	40
1	16	14	37	40			
1	16	13	27	60	16	80	40
1	16	15	41	40			
1	16	14	30	44	16	65	40

Table 8.3: Risultati dell'esecuzione dei due modelli sulle istanze di test da 51 a 75.

Parametri		ASP			MiniZinc		
K	N	nUS	avg(US)	nSan	nUS	max(US)	nSan
2	8	0	0	120	0	0	84
2	9	1	59	88	0	0	88
2	9	2	37	136	0	0	88
2	9	0	0	88	0	0	88
2	9	1	84	144	0	0	84
2	9	0	0	96	0	0	84
2	9	1	85	80	0	0	80
2	9	1	6	88	0	0	80
2	10	1	6	88	0	0	84
2	10	2	75	80	0	0	88
2	11	3	49	96	0	0	88
2	11	3	45	104	0	0	88
2	11	4	25	104	0	0	80
2	12	5	69	104	0	0	84
2	12	4	59	104	0	0	80
2	12	3	45	80	0	0	80
2	12	4	44	96	5	6	84
2	12	5	32	96	3	7	84
2	12	3	52	96			
2	13	5	45	88			
2	13	6	42	104	5	10	80
2	13	6	33	88	0	0	96
2	13	5	68	112	0	0	84
2	13	4	49	104	7	2	84
2	13	5	63	96			

Table 8.4: Risultati dell'esecuzione dei due modelli sulle istanze di test da 76 a 100.

Parametri		ASP			MiniZinc		
K	N	nUS	avg(US)	nSan	nUS	max(US)	nSan
2	14	11	45	96			
2	14						
2	14	8	51	96	6	16	84
2	14	7	66	128	8	15	84
2	14	10	53	120			
2	14	6	52	96	6	15	84
2	14	7	55	88			
2	14						
2	15	10	47	104	3	4	80
2	15	7	24	80			
2	15	10	43	112			
2	15	9	39	120	9	32	88
2	15	9	35	96	8	31	88
2	15	10	52	176			
2	15	9	56	136			
2	15	8	41	88			
2	16	13	27	88	11	32	84
2	16	10	63	104	7	33	84
2	16	13	48	160			
2	16	9	49	96			
2	16	13	43	96			
2	16	8	43	104			
2	16	14	23	88			
2	16	10	51	120			
2	16	7	61	104			

References

[1] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[2] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.

[3] Andrea Schaerf. A survey of automated timetabling. *Artif. Intell. Rev.*, 13(2):87–127, 1999.