Sessione d'esami

Relazione di progetto

Tobia Cargnello

Automated Reasoning

Indice

1 Introduzione **2**
1.1 Consegna 2
1.2 Metodo 2
1.3 Strumenti utilizzati 3

2 Implementazione **3**
2.1 Minizinc 3
2.1.1 Input 3
2.1.2 Codice sorgente 4
2.1.3 Output 7
2.2 Answer Set Programming 8
2.2.1 Input 8
2.2.2 Codice sorgente 8
2.2.3 Output 11

3 Risultati sperimentali **12**
3.1 Creazione di istanze di test 12
3.2 Risultati 14

4 Conclusioni **15**

1 Introduzione

La seguente relazione si riferisce al progetto del corso di Ragionamento Automatico (Automated Reasoning), di seguito ne viene riportata la consegna.

1.1 Consegna

Si consideri un insieme di n studenti $s_1..s_n$. Ogni studente seleziona (ad esempio con un vettore di bit) alcuni esami da un insieme di esami E. Il problema è quello di schedulare gli esami in t giorni consecutivi partendo da un lunedì e lasciando liberi il sabato e la domenica. I vincoli sono i seguenti:

- vi sono due aule;
- ogni giorno, in un'aula vi è al massimo un esame al mattino e uno al pomeriggio (due slot giornalieri per aula);
- ogni studente deve poter partecipare ad almeno 3 esami (sempre che ne abbia selezionati 3 o più, se ne ha selezionati 2 o 1 i limiti si abbassano);
- devono passare almeno due giorni pieni vuoti tra un esame ed il successivo.

I compiti da portare a termine per il progetto sono i seguenti:

- si codifichi il problema;
- si minimizzi t che garantisca la soluzione;
- si trovi la soluzione che massimizza il numero di esami che possono essere dati (anche più di 3 per studente).

1.2 Metodo

Per modellare il problema sono state fatte le seguenti assunzioni:

- uno studente non ha, in principio, un limite sul numero di esami che può scegliere di prenotare (tra quelli disponibili);
- la sessione di esami considerata non prevede appelli multipli, ciascun esame viene svolto in un solo giorno.

I giorni vengono rappresentati tramite numeri interi crescenti a partire da 1, dove 1 identifica il lunedì. Per identificare a quale giorno della settimana un numero corrisponde, basterà calcolare il modulo 7 del numero in questione; dato n il numero di un giorno, si ha che:

- $n = 1 \pmod 7$ il giorno è lunedì;
- $n = 2 \pmod 7$ il giorno è martedì;
- $n = 3 \pmod 7$ il giorno è mercoledì;

- $n = 4 \pmod 7$ il giorno è giovedì;
- $n = 5 \pmod 7$ il giorno è venerdì;
- $n = 6 \pmod 7$ il giorno è sabato;
- $n = 0 \pmod 7$ il giorno è domenica.

Il numero massimo dei giorni è stato fissato a 60, questo per indicare che la sessione d'esami può avere una durata massima di due mesi (sabati e domeniche comprese).

Si diversificano esami "prenotati", esami "prenotabili" ed esami "accettabili". Gli esami prenotati sono tutti quelli che uno studente ha intenzione di svolgere, gli esami prenotabili sono gli esami che uno studente può effettivamente svolgere una volta organizzato il calendario degli esami. Gli esami accettabili comprendono gli esami prenotabili per uno studente e gli esami che, pur non essendo stati prenotati da uno studente, possono essere comunque svolti poiché soddisfano i vincoli definiti. La soluzione calcolata, ovvero il numero di giorni impiegati per programmare gli esami, comprende nel conteggio anche i sabati e le domeniche (giorni nei quali però non viene svolto alcun esame).

1.3 Strumenti utilizzati

Per la realizzazione del progetto sono stati utilizzati i seguenti strumenti:

- Minizinc to FlatZinc converter 2.5.3;
- Clingo 5.2.2;
- Python 3.10.4;
- Visual Studio Code (Editor).

2 Implementazione

Viene descritta in questa sezione la modellazione del problema nei linguaggi Minizinc e Answer Set Programming (ASP) e le scelte d'implementazione nei rispettivi linguaggi.

2.1 Minizinc

2.1.1 Input

Per modellare il problema in Minizinc, si è scelto di rappresentare esami, studenti, aule e slot temporali attraverso il tipo di dato `enum`. Il modo in cui esami, studenti, aule e slot temporali vengono identificati è, in generale, a discrezione dell'utente che definisce questi dati in input. Tali dati sono definiti in un file di input. L'input del programma viene definito in un file separato contenente la

lista degli esami, degli studenti, delle aule disponibili, degli slot temporali disponibili e le prenotazioni degli esami da parte degli studenti. Le prenotazioni degli esami da parte degli studenti sono rappresentati tramite una matrice `booking`; sulle colonne troviamo gli esami, mentre sulle righe gli studenti (ordinati secondo l'ordine dato in input). In corrispondenza di ogni cella esame-studente, vi è un valore 1 se l'esame è stato prenotato dallo studente, 0 altrimenti (Listato 2.1).

```
enum EXAMS;
enum STUDENTS;
enum ROOMS;
enum SLOTS;

array[STUDENTS,EXAMS] of 0..1: booking;
```

Listato 2.1: Input del modello

2.1.2 Codice sorgente

Per rappresentare il calendario vengono utilizzati 3 vettori; il primo denominato `exam_days` per ogni esame associa il giorno in cui questo viene programmato, il secondo denominato `exam_room` e il terzo denominato `exam_slot` associano rispettivamente un'aula e uno slot temporale ad ogni esame. La variabile rappresentate il numero totale dei giorni è data dal numero identificante l'ultimo in cui un esame viene programmato, che sarà il numero massimo utilizzato (Listato 2.2).

```
array[EXAMS] of var 0..60: exams_days;
array[EXAMS] of var ROOMS: exams_rooms;
array[EXAMS] of var SLOTS: exams_slots;

array[STUDENTS,EXAMS] of var 0..1: acceptable;

var int: total_days = max(exams_days);
```

Listato 2.2: Definizione del calendario

I numeri identificativi dei giorni partono dal numero 1, che identifica un lunedì. Pertanto i giorni non possono essere identificati con numeri minori di 1 (Listato 2.3).

```
constraint forall(i in EXAMS)(
    exams_days[i]>=1
);
```

Listato 2.3: Vincolo sui numeri identificanti i giorni

Per essere prenotabili da uno studente, gli esami devono avere una distanza minima di due giorni l'uno dall'altro. Ciò significa che per poter dare un esame successivo ad uno già dato, lo studente deve aspettare due giorni interi (Listato 2.4).

```
predicate at_least_2_days_apart(EXAMS: i, EXAMS: j) =
    exams_days[j] > exams_days[i]+2;

predicate at_least_2_days_apart(EXAMS: i, EXAMS: j, EXAMS
    : k) =
    exams_days[k] > exams_days[j] + 2 /\
    exams_days[j] > exams_days[i]+2;
```

Listato 2.4: Definizione del predicato per il controllo della distanza tra due giorni

Secondo i vincoli definiti dalla consegna, due esami non possono essere svolti nello stesso giorno, nella stessa aula e nello stesso slot temporale (Listato 2.5).

```
constraint forall(i,j in EXAMS where i != j)(
    if exams_days[i]=exams_days[j] /\ exams_rooms[i]=
        exams_rooms[j]
    then exams_slots[i]!=exams_slots[j]
    else true
    endif
);
```

Listato 2.5: Due esami non possono essere svolti contemporaneamente nello stesso luogo

Gli esami non possono essere programmati nei sabati e nelle domeniche, in termini di numeri identificanti i giorni ciò significa che tutti i giorni identificati con numeri pari a 0 o 6 considerando il modulo 7 non possono essere scelti come giorni per programmare gli esami (Listato 2.6).

```
constraint forall(e in EXAMS)(
    'mod'(exams_days[e],7) != 6 /\
    'mod'(exams_days[e],7) != 0
);
```

Listato 2.6: Non ci possono essere esami al sabato e alla domenica

Uno studente che abbia prenotato più di 3 esami, deve poter essere in grado di svolgerne almeno 3, ciò significa che almeno 3 degli esami prenotati da uno studente devono essere prenotabili per quello studente (e quindi anche accettabili). Uno studente che abbia prenotato 2 esami, deve poter essere in grado di svolgerli entrambi, ciò significa che i 2 esami prenotati dallo uno studente devono essere prenotabili per quello studente (e quindi anche accettabili). Uno studente che abbia prenotato soltanto un esame, potrà svolgerlo indipendentemente dal giorno in cui l'esame è stato programmato (Listato 2.7).

```
constraint forall(s in STUDENTS)(
    if count(booking[s,..],1)>=3
    then exists(i,j,k in EXAMS where booking[s,j] = 1 /\
        booking[s,k] = 1 /\ booking[s,i]=1)
            (at_least_2_days_apart(i,j,k) /\ acceptable[s
                ,i]=1 /\ acceptable[s,j]=1 /\ acceptable[s
                ,k] = 1)
    else
        if count(booking[s,..],1)=2
        then exists(i,j in EXAMS where booking[s,j] = 1
            /\ booking[s,i]=1)
            (at_least_2_days_apart(i,j) /\ acceptable[s,i
                ]=1 /\ acceptable[s,j]=1)
        else true
        endif
    endif
);
```

Listato 2.7: Vincolo sul numero di esami effettivamente prenotabili da uno studente

Un esame, per essere accettabile per uno studente, deve avere una distanza di almeno 2 giorni da tutti gli esami prenotabili per uno studente. Inoltre, un esame accettabile per uno studente lo è quando questo ha una distanza di almeno due giorni da ogni esame già definito come accettabile per uno studente, che sia un esame prenotabile o meno (Listato 2.8).

```
constraint forall(s in STUDENTS, e1 in EXAMS)(
    if forall(e2 in EXAMS where e1!=e2 /\ acceptable[s,e2
       ]=1)(
            abs(exams_days[e2]-exams_days[e1])>2
            )
    then acceptable[s,e1]=1
    else acceptable[s,e1]=0
    endif
    );

constraint forall(s in STUDENTS, e1 in EXAMS)(
    not exists(e2 in EXAMS where e1!=e2)(
        abs(exams_days[e1]-exams_days[e2])<=2 /\
           acceptable[s,e1] = 1 /\ acceptable[s,e2]=1
    )
);
```

Listato 2.8: Vincoli sugli esami accettabili

La soluzione prevede di minimizzare il numero di giorni in cui gli esami vengono programmati e, successivamente, massimizzare il numero di esami che uno studente, in generale, può fare. Questo numero equivale al numero di esami accettabili dato un calendario. Per minimizzare il numero di giorni totali della sessione basta minimizzare il massimo numero identificante un giorno. Questo numero è calcolato nella variabile `total_days`. Una volta minimizzato questo valore, il massimo numero di esami che uno studente può fare è dato dal numero di esami accettabili. Questo può essere anche maggiore di 3, a seconda del calendario, delle prenotazioni e degli studenti in una sessione (Listato 2.9).

```
solve:: int_search(
    exams_days,
    most_constrained,
    indomain_min,
    complete
)
minimize(total_days);
```

Listato 2.9: Soluzione

2.1.3 Output

L'output del programma consiste di 4 matrici. Le prime 3 mostrano rispettivamente il giorno, l'aula e lo slot in cui ogni esame viene svolto. La quarta matrice ha su ogni riga gli studenti, su ogni colonna gli esami. Essa mostra, per ogni studente, gli esami accettabili, ovvero quelli che uno studente può effettivamente

svolgere data la programmazione, che rientrino o no tra gli esami prenotati da ciascuno studente. Un esempio di output è presentato in Figura 1.

```
GIORNO
1       analisi_matematica
1       programmazione_e_laboratorio
4       fisica
1       matematica_discreta

AULA
a1       analisi_matematica
a1       programmazione_e_laboratorio
a1       fisica
a2       matematica_discreta

SLOT
p       analisi_matematica
m       programmazione_e_laboratorio
m       fisica
m       matematica_discreta

ACCEPTABLE
1       0       1       0
1       0       1       0
```

Figura 1: Esempi output programma Minizinc

2.2 Answer Set Programming

2.2.1 Input

I dati in input vengono specificati in un file a parte e non fanno parte del modello. In particolare, nel file di input viene specificato l'intervallo di numeri identificanti i giorni. Gli studenti vengono identificati con il predicato `student(S)`, gli esami con il predicato `exam(E)`, le aule con il predicato `room(R)`, gli slot temporali con il predicato `slot(L)`. Le prenotazioni degli esami da parte degli studenti vengono definite con il predicato `book(S,E)`, dove `S` indica uno studente ed `E` indica un esame.

2.2.2 Codice sorgente

Vengono definiti i predicati `exam_day(E,D)`, `exam_room(E,R)` e `exam_slot(E,L)` che indicano rispettivamente il giorno, l'aula e lo slot temporale in cui un esame viene programmato. Ciascun esame può essere programmato in un solo giorno, in una sola aula, in un solo slot temporale, come stabilito nelle assunzioni (1.2). Nella modellazione sono stati utilizzati dei vincoli sulle cardinalità (Listato 2.10).

```
1 {exam_day(E,D): day(D)} 1      :- exam(E).
1 {exam_room(E,R): room(R)} 1    :- exam(E).
1 {exam_slot(E,L): slot(L)} 1    :- exam(E).
```

Listato 2.10: programmazione di un esame

I giorni non possono essere identificati con numeri minori di 1, pertanto si impone il vincolo che un giorno, per essere tale, deve essere identificato da un valore che sia maggiore di 1 (Listato 2.11).

```
:- day(X), X<1.
```

Listato 2.11: Vincolo sul numero identificativo di un giorno

Due esami non possono essere programmati nello stesso giorno, nella stessa aula e nello stesso slot temporale, pertanto si impone il vincolo seguente (Listato 2.12).

```
:- exam_day(E1,D), exam_day(E2,D), day(D),
   exam(E1), exam(E2),
   exam_room(E1,R), exam_room(E2,R), room(R),
   exam_slot(E1,L), exam_slot(E2,L), slot(L),
   E1 != E2.
```

Listato 2.12: Vincolo sulla programmazione degli esami

Gli esami non possono essere programmati nei sabati e nelle domeniche, come nel caso di Minizinc anche qui si impone che i giorni identificati con numeri uguali a 0 o 6 modulo 7 non possano essere utilizzati per programmare gli esami (Listato 2.13).

```
:- exam_day(E,D), day(D), exam(E), D\7=6.
:- exam_day(E,D), day(D), exam(E), D\7=0.
```

Listato 2.13: Non ci possono essere esami il sabato e la domenica

Il numero minimo di esami che uno studente deve essere in grado di poter fare (almeno 3 per studenti che abbiano prenotato 3 o più esami, 2 per studenti che abbiano prenotato 2 esami e 1 per studenti che abbiano prenotato un solo esame) viene indicato attraverso vincoli sulla cardinalità. In generale il numero di esami prenotabili da uno studente può essere al massimo pari al numero di esami totali disponibili, e come minimo 3, 2 o 1 a seconda del numero di esami prenotati da uno studente (Listato 2.14).

```
3 {bookable(S,E): exam(E), book(S,E)} N :- student(S),
    total_exams(C), exams_booked(S,N), N>2.

2 {bookable(S,E): exam(E), book(S,E)} N :- student(S),
    total_exams(C), exams_booked(S,N), N>1.

1 {bookable(S,E): exam(E), book(S,E)} N :- student(S),
    total_exams(C), exams_booked(S,N), N>0.
```

Listato 2.14: Vincolo sugli esami effettivamente prenotabili dagli studenti

Uno studente non può svolgere degli esami che non siano a distanza di almeno due giorni interi l'uno dall'altro, perciò si impone il vincolo sul predicato `bookable(S,E)`, dove `S` e `E` indicano rispettivamente uno studente e un esame (Listato 2.15).

```
:- student(S),
    exam(E1), exam(E2),
    bookable(S,E1), bookable(S,E2),
    day(D1), day(D2),
    exam_day(E1,D1), exam_day(E2,D2),
    E1!=E2,
    |D1-D2|<=2.
```

Listato 2.15: Uno studente non può svolgere due esami che sono a distanza minore di due giorni

Il numero di esami accettabili per uno studente è, in generale, come minimo pari al numero di esami prenotabili per uno studente e al massimo pari al numero di esami totali disponibili, pertanto anche qui si sfruttano i vincoli sulle cardinalità (Listato 2.16).

```
N {acceptable(S,E): exam(E)} C :- student(S), total_exams
    (C), exams_bookable(S,N).
```

Listato 2.16: Esami accettabili

Per essere accettabile, un esame deve avere una distanza di due giorni interi da ogni esame prenotabile per uno studente, inoltre non ci possono essere due esami accettabili che però non abbiamo due giorni interi di distanza l'uno dall'altro (Listato 2.17).

```
:- student(S),
    exam(E1), exam(E2),
    acceptable(S,E1), acceptable(S,E2),
    day(D1), day(D2),
    exam_day(E1,D1), exam_day(E2,D2),
    E1!=E2,
    |D1-D2|<=2.

:- student(S),
    exam(E1), exam(E2),
    acceptable(S,E1), bookable(S,E2),
    day(D1), day(D2),
    exam_day(E1,D1), exam_day(E2,D2),
    E1!=E2,
    |D1-D2|<=2.
```

Listato 2.17: Vincoli sugli esami accettabili

La soluzione prevede di minimizzare il numero di giorni impiegati per programmare gli esami e, una volta minimizzato questo valore, massimizzare il numero di esami accettabili per ogni studente. Si calcola così il massimo numero che identifica un giorno e si chiede di minimizzare tale numero. Si calcola poi il numero di esami accettabili per uno studente e si chiede di massimizzarlo (Listato 2.18).

```
max_day(M) :- M = #max{D : exam_day(E,D), day(D)}, exam(E
    ).

n(T) :- T = #sum{M: max_day(M)}.

#minimize {T:n(T)}.

#maximize {N: exams_acceptable(S,N)}.
```

Listato 2.18: Calcolo della soluzione

2.2.3 Output

L'output del programma in ASP si compone dei predicati `exam_day(E,D)`, `exam_room(E,R)`, `exam_slot(E,L)` e `acceptable(S,E)` che indicano rispettivamente il giorno, l'aula e lo slot temporale in cui ogni esame è stato programmato e gli esami accettabili per ogni studente. Il codice per l'output del programma è descritto nel Listato 2.19, un esempio di output del programma è rappresentato in Figura 2.19.

```
#show exam_day/2.
#show exam_room/2.
#show exam_slot/2.
#show acceptable/2.
```

Listato 2.19: Calcolo della soluzione

```
Answer: 25
acceptable(0,programmazione_e_laboratorio) acceptable(0,fisica) acceptable(0,matematica_discreta) a
cceptable(1,programmazione_e_laboratorio) acceptable(1,fisica) acceptable(1,matematica_discreta) ex
am_slot(analisi_matematica,m) exam_room(analisi_matematica,a1) exam_room(programmazione_e_laborator
io,a1) exam_room(fisica,a2) exam_slot(matematica_discreta,m) exam_room(matematica_discreta,a1) exam
_slot(programmazione_e_laboratorio,p) exam_slot(fisica,p)
Optimization: 16
OPTIMUM FOUND

Models       : 25
  Optimum    : yes
Optimization : 16
Calls        : 1
Time         : 0.258s (Solving: 0.13s 1st Model: 0.00s Unsat: 0.01s)
CPU Time     : 0.258s
```

Figura 2: Esempi output programma ASP

3 Risultati sperimentali

Tutti gli esperimenti sono stati condotti su una macchina con le seguenti specifiche:

- CPU: Intel i7-4720HQ, 3.600GHz
- GPU: Nvidia Geforce GTX 950M
- RAM: 16 GB DDR3L-1600
- SO: Ubuntu 22.04 (LTS)

Tutti i test sono stati condotti assegnando un tempo limite di 5 minuti all'esecuzione di ciascun programma su ogni istanza di test. Sono state create in tutto 30 istanze di test: 10 istanze con 10 studenti e 5 esami, 10 istanze con 500 studenti e 19 esami e 10 istanze con 100 studenti e 51 esami.

3.1 Creazione di istanze di test

Per la generazione delle istanze di test è stato definito uno programma scritto in Python (`test_generator.py`). Il programma contiene una lista di esami ed accetta in input un numero intero indicate il numero di studenti. Ogni studente viene indicato con un numero intero partendo da 0. Per ogni studente viene generato un vettore dove in corrispondenza di ogni colonna vi è un numero pseudo random tra 0 e 1 che indica se quello studente ha prenotato quell'esame (valore

1) oppure no (valore 0). Lo script genera un file di input con i valori specificati sia per il modello in Minizinc che per il modello in ASP. Nel programma vi è anche definita la lista delle aule, che in questo caso sono 2 e sono denominate "a1" e "a2". Vi è anche la lista degli slot temporali, che sono anch'essi 2 e sono indicati con "m" (mattina) e "p" (pomeriggio).

Per effettuare i test di tutte le istanze è stato usato lo script Python seguente (`tester.py`):

```
import os
for i in range(0,10):
    str_lp_simple = (f"clingo --outf=1 --time-limit=300
        exams_scheduler.lp testsSimple/data{i}.lp")
    str_dzn_simple = (f"minizinc --solver Chuffed -f --
        time-limit 300000 --output-time exams_scheduler.
        mzn testsSimple/data{i}.dzn")
    str_lp_medium = (f"clingo --outf=1 --time-limit=300
        exams_scheduler.lp /testsMedium/data{i}.lp")
    str_dzn_medium = (f"minizinc --solver Chuffed -f --
        time-limit 300000 --output-time exams_scheduler.
        mzn testsMedium/data{i}.dzn")
    str_lp_large = (f"clingo --outf=1 --time-limit=300
        exams_scheduler.lp testsLarge/data{i}.lp")
    str_dzn_large = (f"minizinc --solver Chuffed -f --
        time-limit 300000 --output-time exams_scheduler.
        mzn testsLarge/data{i}.dzn")
    os.system(str_dzn_simple + " > " + f"out{i}
        _minizinc_simple.txt")
    os.system(str_lp_simple + " > " + f"out{i}_asp_simple
        .txt")
```

Listato 3.1: Script per effettuare i test su tutte le istanze

Per avviare i programmi `test_generator.py` e `tester.py` basterà eseguirli da terminale utilizzando Python, ad esempio con i comandi:
`Python3 test_generator.py`
`Python3 tester.py`
Il primo genererà una batteria di 10 istanze. Le liste degli esami, delle aule e degli slot temporali possono essere modificate a piacimento. Il secondo eseguirà i modelli su ogni istanza creata, generando un file con l'output di ciascun modello per ogni istanza. È necessario specificare il percorso corretto delle istanza da testare all'interno dello script.

3.2 Risultati

Per i test del modello in Minizinc è stato usato il solver **Chuffed**, poiché utilizzando il solver **Gecode** il modello non era in grado di raggiungere una soluzione entro lo scadere del tempo già con istanze di medie dimensioni. I risultati sono riportati nella tabella seguente.

STUDENTI	ESAMI	GIORNI TOTALI MINIZINC	TEMPO MINIZINC (s)	GIORNI TOTALI ASP	TEMPO ASP (s)
10	5	8	0.19	8	0.267
10	5	11	0.17	11	0.266
10	5	11	0.2	11	0.245
10	5	11	0.19	11	0.283
10	5	8	0.2	8	0.229
10	5	11	0.15	11	0.229
10	5	8	0.17	8	0.217
10	5	8	0.19	8	0.22
10	5	11	0.23	11	0.197
10	5	11	0.2	11	0.186
500	19	11	79.77	11	297.961
500	19	11	79.03	11	298.237
500	19	15	224.4	15	time out
500	19	11	64.72	11	time out
500	19	11	62.92	11	time out
500	19	11	65.28	11	time out
500	19	11	87.08	15	time out
500	19	11	113.22	11	time out
500	19	11	96.14	11	time out
500	19	11	78.37	15	time out
100	51	-	time out	60	time out
100	51	-	time out	60	time out
100	51	-	time out	57	time out
100	51	-	time out	60	time out
100	51	-	time out	60	time out
100	51	-	time out	60	time out
100	51	-	time out	60	time out
100	51	-	time out	60	time out
100	51	-	time out	60	time out
100	51	-	time out	60	time out

Figura 3: Risultati sperimentali

Sulle istanze di test più piccole si nota che il modello in Minizinc è comparabile rispetto al modello in ASP per quanto riguarda il tempo di esecuzione. Su istanze più grandi con un numero di esami poco più alto ed un numero considerevole di studenti, si nota come il modello in Minizinc riesca a trovare una soluzione entro il tempo limite, mentre il modello in ASP non riesca a calcolare la soluzione ottimale entro il tempo previsto. Su istanze dove invece si ha un cospiquo numero di esami, il modello in Minizinc non riesce a trovare una soluzione entro il tempo limite come pure il modello in ASP, che però restituisce una soluzione con il numero di giorni massimo (in questo caso 60).

4 Conclusioni

Dai risultati sperimentali si può quindi concludere che su istanze piccole del problema, il modello in Minizinc e il modello in ASP hanno prestazioni simili, quindi non vi sono grosse differenze sulla scelta di utilizzo di un modello o dell'altro. Su istanze di medie dimensioni, con un numero limitato di esami ed un numero considerevole di studenti, il modello in Minizinc risulta preferibile in quanto in grado di trovare una soluzione entro il tempo limite. In presenza però di un numero di esami più consistente e di un più limitato numero di studenti, il modello in ASP sembra, in principio, preferibile poiché in grado di fornire una soluzione, anche se non ottimale, entro lo scadere del tempo, mentre il modello in Minizinc non riesce a fornire alcuna soluzione entro il tempo limite. Un possibile limite alle prestazioni del modello in ASP può essere il file di dati in input che cresce rapidamente al crescere del numero di studenti ed esami, aumentando il tempo richiesto per la lettura. In particolare al crescere del numero di esami, vi sono molte occorrenze del predicato `book(S,E)`. Una possibile soluzione al problema della grandezza del file in input per il programma in ASP portebbe essere quella di limitare il numero di esami che uno studente può scegliere di fare, in tal caso si avrebbero molte meno occorrenze del predicato `book(S,E)` in input e il file avrebbe dimensioni più contenute.