Catholic University of Louvain

Project 4 : Modeling

LINGI2365 - Constraint Programming

Auteurs :
Vanwelde Romain (3143-10-00)
Crochelet Martin (2236-10-00)

Groupe 7

Superviseurs :
Pr. Yves Deville
François Aubry

5 avril 2014

Table des matières

1 Louvain-La-Neuve Golfer Problem **1**

1.1 Explain which symmetries can arise for this problem. 1

1.2 Describe two possible models for this problem. 1

1.2.1 explain your models in detail 1

1.2.2 explain how you can modify your models to take symmetries into account 1

1.3 Implement both models (considering symmetries) in Comet. 1

1.4 Explain which variable / value ordering heuristics you use with each model. 1

1.5 What is the theoretical maximum number of week in a schedule ? 1

1.6 Indicate the maximum number of weeks you could identify in a reasonable time limit using both models. 1

1.7 Indicate for each model, for each number of weeks the time needed to find a solution, the number of failures and the number of choices. Explain the results, do they correspond to what you would have expected ? 1

2 The Time Tabling Problem **1**

2.1 Explain which symmetries arise in this problem. 1

2.2 Design an efficient model for this problem. 1

2.3 Explain your model for this problem and explain how it handles symmetries. 2

2.4 Implement your model in Comet. 3

2.5 To solve this problem you will need a search procedure that is more efficient than a simple label. 4

2.6 Test your model on each of the instances provided on the iCampus site. Indicate for each instance the time needed to solve it, the number of failures and the number of choices. 4

1 Louvain-La-Neuve Golfer Problem

1.1 Explain which symmetries can arise for this problem.

1.2 Describe two possible models for this problem.

1.2.1 explain your models in detail

1.2.2 explain how you can modify your models to take symmetries into account

1.3 Implement both models (considering symmetries) in Comet.

1.4 Explain which variable / value ordering heuristics you use with each model.

1.5 What is the theoretical maximum number of week in a schedule ?

1.6 Indicate the maximum number of weeks you could identify in a reasonable time limit using both models.

1.7 Indicate for each model, for each number of weeks the time needed to find a solution, the number of failures and the number of choices. Explain the results, do they correspond to what you would have expected ?

2 The Time Tabling Problem

2.1 Explain which symmetries arise in this problem.

The first symmetry we find is in relation with timeslots. Indeed, we can easily invert all the course given at specific timeslot with all the courses of another timeslot.

A second symmetry could be the attribution of rooms that are exactly the same (same features, and same capacity). Since we assume that this kind of symmetry doesn't happen often, and that it's something quite tricky to express in our model, we didn't implement it in our model.

2.2 Design an efficient model for this problem.

```
...

// create data variables
int roomsize[rroom];                     // roomsize
int atends[rstudent,revent];             // attends
int roomequipment[rroom,rfeature];       // room equipment
int eventrequirement[revent,rfeature];  // course needed equipment

int nstudentattends[revent] = 0;        // student # enroled in the course


...


// model variables
tuple triple {int a1;int a2;int a3;}

set{triple} Triples();
forall (i in 1..nroom, j in timeslot)
   Triples.insert(triple(i,j,(i-1)*ntime + j));

Table<CP>roomslot(all(e in Triples) e.a1, all(e in Triples) e.a2, all(e in
    Triples) e.a3);


var<CP>{int} lectureslot[revent](cp, timeslot);
var<CP>{int} lectureroom[revent](cp, rroom);
var<CP>{int} roomslotid[revent](cp, 1..nroom*ntime);
```

time_tabling_model.co

2.3 Explain your model for this problem and explain how it handles symmetries.

In the model, the data variables contains information extract from the inputfile. We added one data variable (`nstudentattends`) which will store the number of students which assists to different courses.

The model is composed of 3 model variables, which are `lectureslot`, `lectureroom` and `roomslotid`. `lectureslot` gives, for each event, all the possible timeslots for this event. `lectureroom` gives, for each event, all the possible rooms for this event. Finally, `roomslotid` gives an ID for each possible combination of room/timeslot.

```
solve<cp> {
   ...
} using {
   label(lectureslot[1],1);

   forall(e in revent) by (lectureroom[e].getSize())
      tryall<cp>(r in rroom : lectureroom[e].memberOf(r)) by (r)
         label(lectureroom[e],r);

   forall(e in revent) by (lectureslot[e].getSize())
      tryall<cp>(t in timeslot : lectureslot[e].memberOf(t)) by (t)
         label(lectureslot[e],t);
}
```

time_tabling_symmetry.co

The symmetry concerning the time is handeled in the solver, we can arbitrary fix the first timeslot to any course. Then, we continue to handle it by trying to bound `lectureslot` by growing slot ID. An event will then be assign to a slot which already contains some events, or the first which has no events yet, but it will never visit events with higher ID (if the problem is solvable).

2.4 Implement your model in Comet.

Here are the constraints implemented in Comet. All others important part are already in the previous sections.

```
solve<cp> {
   // A student can attend only one lecture at any time slot
   forall (s in rstudent){
      cp.post(alldifferent(all (e in revent : atends[s,e] == 1) lectureslot
          [e]),onBounds);
   }

   // compute roomslot's id
   forall (e in revent){
      cp.post((lectureroom[e]-1) * ntime + lectureslot[e] == roomslotid[e])
          ;
   }

   // A room can hold only one lecture at any time slot
   cp.post(alldifferent(all (e in revent) roomslotid[e]),onDomains);


   // Table constraint
   forall (e in revent){
      cp.post(table(lectureroom[e],lectureslot[e],roomslotid[e],roomslot));
   }

   // A lecture can take place only in a room having the required features
   forall(e in revent) {
      forall (f in rfeature) {
         cp.post(eventrequirement[e,f] <= roomequipment[lectureroom[e],f]);
      }
   }

   // A lecture can take place only in a room big enough to hold all the
       students that need to attend
   forall (e in revent){
      cp.post(nstudentattends[e]<=roomsize[lectureroom[e]]);
   }

} using {
   ...
}
```

time_tabling_constraint.co

2.5 To solve this problem you will need a search procedure that is more efficient than a simple label.

We first implemented our model by binding variables which have few rooms left, then by binding variables which have few timeslots left. We decided to do it in this way (and not first the timeslots then the rooms) after some tests which were showing that the second solution were far away lower than the first one. This is probably because finding an appropriate room is more difficult in this problem than finding a good timeslot. The code is already given in previous section, and the results of the tests are in the next one.

2.6 Test your model on each of the instances provided on the iCampus site. Indicate for each instance the time needed to solve it, the number of failures and the number of choices.

File ID	Time	#Failures	#Choices
#1	3685	1	800
#2	3535	0	800
#3	2671	368	1041
#4	307397	259291	140520
#5	Out Of Time	Out Of Time	Out Of Time
#6	4044	3	700
#7	3934	0	700
#8	3692	0	800
#9	5222	0	880
#10	3548	0	800
#11	4375	0	800
#12	3671	0	800
#13	4017	0	800
#14	4530	0	700
#15	4085	0	700
#16	Out Of Time	Out Of Time	Out Of Time
#17	3718	0	700
#18	3126	0	800
#19	4186	0	800
#20	4000	1	700