Monash University of Melbourne
Faculty of Information Technology
Semester 1 Assessment 2020

FIT5216 Modelling for Discrete Optimization: Sample with Answers

Time Allowed: 2 Hours
Reading Time: 15 minutes

Authorized materials: Books and calculators are not permitted.

Instructions to Invigilators: One 21 page script. Exam paper may not leave the room.

Instructions to students:
This exam counts for 35% of your final grade. There are 9 pages and 6 questions for a total of 35 marks. Attempt to answer all of the questions. Values are indicated for each question and subquestion — be careful to allocate your time according to the value of each question.

Question 1 [5 marks]

Alex, Brook, Cody, Dusty, and Erin recently found out that all of their birthdays were on the same day, though they are different ages.

On their mutual birthday, they were jabbering away, flapping their gums about their recent discovery. And, lucky me, I was there. Some of the things that I overheard were...

- Dusty said to Brook: ”I'm nine years older than Erin.”
- Erin said to Brook: ”I'm seven years older than Alex.”
- Alex said to Brook: ”Your age is exactly 70% greater than mine.”
- Brook said to Cody: ”Erin is younger than you.”
- Cody said to Dusty: ”The difference between our ages is six years.”
- Cody said to Alex: ”I'm ten years older than you.”
- Cody said to Alex: ”Brook is younger than Dusty.”
- Brook said to Cody: ”The difference between your age and Dusty's is the same as the difference between Dusty's and Erin's.”

Since I knew these people – and how old they were, I knew that they were not telling the whole truth.

After thinking about it, I realized that when one of them spoke to someone older, everything they said was true, but when speaking to someone younger, everything they said was false.

Write a MiniZinc model to determine the ages of each each person assuming they are integers in the range 5..100.

```
var 5..100: Alex;
var 5..100: Brook;
var 5..100: Cody;
var 5..100: Dusty;
var 5..100: Erin;
include "alldifferent.mzn"; % wont be penalized for forgetting this
constraint alldifferent([Alex,Brook,Cody,Dusty,Erin]);
constraint Dusty < Brook <-> Dusty = Erin + 9;
constraint Erin < Brook <-> Erin = Alex + 7;
constraint Alex < Brook <-> Brook * 10 = Alex * 17;
constraint Brook < Cody <-> Erin <Cody;
constraint Cody < Dusty <-> abs(Cody - Dusty) = 6;
constraint Cody < Alex <-> Cody = Alex + 10;
constraint Cody < Alex <-> Brook < Dusty;
constraint Brook <Cody <-> abs(Cody - Dusty) = abs(Dusty - Erin);
```

Question 2 [5 marks]

There are many counting global constraints of interest in combinatorial problems. One of the simplest is `count_eq` which enforces that c is the number of times that a position in the x array takes the value y. It is defined as

```
    predicate count_eq(array[int] of var int: x, var int: y, var int: c) =
        c = sum(i in index_set(x)) ( x[i] == y );
```

In this question we examine some other counting globals. The combinatorial constraint

```
common(var int: n1, var int: n2,
       array[int] of var int: c1, array[int] of var int: c2)
```

holds if $n1$ is the number of positions in the array $c1$ taking a value in the array $c2$, and $n2$ is the number of positions in the array $c2$ taking a value in the array $c1$. For example

- `common(3,4,[1,9,1,5],[2,1,9,9,6,9])` holds since three positions in [1,9,1,5] are in the set {1,2,6,9} of values of [2,1,9,9,6,9], and similarly 4 values in [2,1,9,9,6,9] are in the set $\{1, 5, 9\}$.
- `common(4,1,[1,1,1,1],[2,1,9,9,6,9])` holds

(a) Give a predicate definition for `common`. [3 marks].

```
    predicate common(var int: n1, var int: n2,
           array[int] of var int: c1, array[int] of var int: c2) =
           n1 = sum(i in index_set(c1))
                   (exists(j in index_set(c2))(c1[i] = c2[j])) /\
           n2 = sum(i in index_set(c2))
                   (exists(j in index_set(c1))(c2[i] = c1[j]));
```

(b) There is a relationship between $n1$ and $n2$ which always holds in every solution of the `common` constraint. Write down the strongest redundant constraint you can that only mentions $n1$ and $n2$. [1 mark].

```
    constraint n1 = 0 <-> n2 = 0
```

(c) The combinatorial constraint

```
    uses(array[int] of var int: c1, array[int] of var int: c2)
```

holds if the set of values in the array $c2$ is included in the set of values in the array $c1$. Write a definition of `uses` making use of the `common` constraint. [1 mark].

```
predicate uses(array[int] of var int: c1, array[int] of var int: c2) =
              common(_, length(c2), c1, c2);
```

Question 3 [4 marks]

Consider the following FlatZinc output.

```
var -1..3: INT1;
var -1..3: INT2;
var -1..3: INT3;
var -1..3: INT4;
var -3..9: INT5;
var -3..9: INT6;
var -3..9: INT7;
var bool: BOOL1;
var bool: BOOL2;
var bool: BOOL3;
constraint int_times(INT1,INT1,INT5);
constraint int_times(INT2,INT2,INT6);
constraint int_times(INT3,INT3,INT7);
constraint int_lin_eq_imp([1,-1],[INT2,INT5],1,BOOL1);
constraint int_lin_eq_imp([1,-1],[INT3,INT6],1,BOOL2);
constraint int_lin_eq_imp([1,-1],[INT4,INT7],1,BOOL3);
constraint array_bool_or([BOOL1,BOOL2,BOOL3,],true);
```

Give the smallest MiniZinc model you can that would generate this FlatZinc. [4 marks].

```
array[1..4] of var -1..3: x;
constraint exists(i in 1..3)(x[i+1] = x[i]*x[i] + 1);
```

Question 4 [6 marks]

In 1779 Euler proposed the following “36 Officer Problem”: given 6 regiments each with 6 officers of 6 different ranks, is it possible to arrange the 36 officers in a square of 6 × 6, so that no row or column contains a duplicate rank or regiment. The mathematician Gastor proved it impossible in 1901.

We consider a generalization of the problem: given n regiments each with n officers of n different ranks, is it possible to arrange the n^2 officers in a $n \times n$ square so that no row or column contains a duplicate rank or regiment. It turns out for many n this is possible. Here is a solution for $n = 4$, shown as (regiment, rank) pairs. Note how no regiment or rank is duplicated in any row or column and each pair occurs exactly once.

```
(4,1) (3,2) (1,3) (2,4)
(2,2) (1,1) (3,4) (4,3)
(3,3) (4,4) (2,1) (1,2)
(1,4) (2,3) (4,2) (3,1)
```

(a) Write a MiniZinc model that given n determines a solution to the $n \times n$ officer problem, if possible. The data format is simply

```
int: n;
```

For example the data for the example above is

```
n = 4;
```

Ensure that any decision variables you define have tight upper and lower bounds. Note you do not have to provide an output item! [4 marks].

```
int: n;
set of int: REG = 1..n;   % defns not required, but clearer
set of int: RANK = 1..n;
set of int: ROW = 1..n;
set of int: COL = 1..n;
array[ROW,COL] of var REG: reg;
array[ROW,COL] of var RANK: rank;
include "alldifferent.mzn";
constraint forall(r in ROW)(alldifferent(reg[r,..]) /\ alldifferent(rank[r,..]));
constraint forall(c in COL)(alldifferent(reg[..,c]) /\ alldifferent(rank[..,c]));
constraint alldifferent([ reg[r,c]*n+rank[r,c] | r in ROW, c in COL ]);
%% output not required, just for your understanding of answer
output [ "(\(reg[r,c]),\(rank[r,c])) " ++ if c = n then "\n" else "" endif
       | r in ROW, c in COL ]
```

(b) Since the regiments and ranks are interchangeable, their are significant symmetries in the problem. Illustrate a symmetry breaking constraint that could be added to your model to improve the solving behavior. Explain why it is correct! [2 marks].

```
include "var_sqr_sym.mzn";  %% not required
constraint var_sqr_sum(reg);
constraint var_sqr_sym(rank);
```

Alternatively

```
include "value_precede_chain.mzn";  %% not required
constraint value_precede_chain(REG,array1d(reg));
constraint value_precede_chain(RANK,array1d(rank));
```

Question 5 [6 marks]

Consider the following partial model for a nurse rostering problem

```
enum NURSE;
int: m; % number of days
set of int: DAY = 1..m;
enum SHIFT = { day, night, dayoff };
int: o; % nurses required on day shift
int: l; % lower bound for nurses on nightshift
int: u; % upper bound for nurses on nightshift

array[NURSE,DAY] of var SHIFT: x;

constraint forall(n in NURSE, d in 1..m-2)
           ( x[n,d] = night /\ x[n,d+1] = night -> x[n,d+2] = dayoff /\
             x[n,d] = night -> x[n,d+1] != day );
constraint forall(n in NURSE)
           ( x[n,m-1] = night -> x[n,m] != day );
```

The model has constraints to ensure that after two night shifts in a row, a nurse must have a dayoff, and that directly after a night shift, a nurse cannot take a day shift

(a) Write constraints for this model that ensure that on every day there are exactly o nurses assigned to a day shift, and between l and u nurses assigned to a night shift. Write the constraint in the most efficient form you can. [2 marks].

```
constraint forall(d in DAY)
           (global_cardinality_low_up(x[..,d], [day,night], [o,l], [o,u]);
```

(b) After adding correct constraints of the previous section that constrain the number of nurses on particular shifts, you find the model returns UNSATISFIABLE for all the data sets. Explain in a paragraph what process you would use to find the error in the model. [2 marks]. Create a minimal data set that still caused the error to occur. Start turning off constraints until the model became satisfiable. Check which constraints, which when turned off cause the model to be satisfiable, versus what I believe they do. Track down exactly which constraint is in error

(c) Explain where the error is in the above model, and how to correct it. [2 marks].
The problem is that the precedence of /\ is tighter than -> so the first forall doesnt do what its meant to do. Modify it to

```
constraint forall(n in NURSE, d in 1..m-2)
           ( (x[n,d] = night /\ x[n,d+1] = night -> x[n,d+2] = dayoff) /\
             (x[n,d] = night -> x[n,d+1] != day) );
```

Question 6 [9 marks]

We need to crew a fleet of trucks with a set of available staff over multiple weeks. Each staff member has a level of experience on each truck and a weekly salary. Each truck must be crewed by enough people so that their total experience with the vehicle reaches a required level. A person can only crew one truck in a week. Each truck must be crewed by at least 1 and at most maxcrew people. The aim is to minimize the total salary paid, where we pay the staff for each week that they work on some truck.

Data for the problem is defined as follows:

```
enum TRUCK;
enum STAFF;
int: n_weeks;
set of int: WEEK = 1..n_weeks;
array[STAFF,TRUCK] of int: experience;
array[STAFF] of int: salary;
int: required_experience;
int: maxcrew;
```

For example the data set

```
TRUCK = { VOLVO, BENZ };
STAFF  = { Mike, Lena, Sue, Rex };
n_weeks = 3;
experience = [| 3, 3 | 0, 6 | 3, 2| 8, 9 ];
salary = [ 6, 10, 11, 20 ];
required_experience = 4;
maxcrew = 2;
```

A possible schedule for 3 weeks is that the Mike and Sue crew the VOLVO and Lena crews the BENZ in each week.

(a) Write a MiniZinc model to solve the above problem. You dont need to repeat the data definitions above. You should document the model to explain it. [4 marks].

```
array[TRUCK,WEEK] of var set of STAFF: staff;
array[STAFF,WEEK] of var bool: works;
%% Each truck must be crewed by enough experience
constraint forall(t in TRUCK, w in WEEK)
                 (sum(s in staff[t,w])(experience[s,t]) >= required_experience);
%% Each staff memner is allocated to at most one truck
constraint forall(s in STAFF, w in WEEK)
                 (sum(t in TRUCK)(s in staff[t,w]) = works[s,w]);
solve minimize sum(s in STAFF, w in WEEK)
                  (works[s,w]*salary[s]);
```

(b) Actually there are rostering limitations. Every staff member must work, in any period of n consecutive weeks, at least l and at most u weeks in that period. Data for this part of the problem is given as

```
int: l; % minimum working weeks
int: u; % maximum working weeks
int: n; % in each n week period
```

Note if $l = 0, u = 2$ and $n = 3$ the previous solution is invalid, as each of Mike, Sue and Lena work all 3 weeks. A new solution is to replace Mike and Sue with Rex on the VOLVO in week 2 and replace Lena with Rex on the BENZ in week 3. Add to your model of part (a) to express these constraints. [2 marks].

```
%% each staff member can work at most max_working_weeks in any period of n weeks
include "sliding_sum.mzn";
constraint forall(s in STAFF)
                 (sliding_sum(l,u,n,works[s,..]));
&& (b) alternate approach without sliding sum
constraint forall(s in STAFF, w in 1..n_weeks-n+1)
           ( let { var int: worked = sum(v in w..w+n-1)(works[s,v]); } in
             worked >= l /\ worked <= u );
```

(c) In fact each week that a staff member works on a truck increases their experience on that truck by 1. This greater experience means the roster can be more flexible in later weeks, since they are more experienced. Modify your model from part (b) to take this into account. With this addition the schedule can just use Mike on the VOLVO in week 3 since his experience will have grown to 4 from week 1. [3 marks].

```
%% (c)
%% modify the model so that if a staff member works on a truck for one week
%% then their experience grows by 1 for that truck. This makes it easier to
%% reach the experience limit in later weeks.
array[STAFF,TRUCK,WEEK] of var 0..infinity: c_exp;
constraint forall(s in STAFF, t in TRUCK)(c_exp[s,t,1] = experience[s,t]);
constraint forall(w in 1..n_weeks-1, s in STAFF, t in TRUCK)
                 (c_exp[s,t,w+1] = (s in staff[t,w])+c_exp[s,t,w]);
%% Each truck must be crewed by enough experience
constraint forall(t in TRUCK, w in WEEK)
                 (sum(s in staff[t,w])(c_exp[s,t,w]) >= required_experience);
```