Languages and Algorithms for Artificial Intelligence, Module 1 Traditional Project, constraints

Sara Vorabbi - 0001026226 - sara.vorabbi@studio.unibo.it

1 Introduction

The following report aims to explore the solutions implemented for the 2021 Autumn Games . The puzzles are solved using the constraint programming paradigm (CP), in particular the language used is MiniZinc, a free and open-source constraint modeling language. Since the Autumn Games proposed different categories of exercises, several were chosen from each category. The project can be found in this GitHub repository.

2 Exercise 1 - Red and black

Problem: "In a deck of 52 cards there are 26 red cards and 26 black cards. The deck is separated into two bundles: the first of 25 cards, the second of 27. If the first deck contains 12 red cards, how many black cards are there in the second deck?"

Code:

```
int: cards = 52;
int: deck_1 = 25;
int: deck_2 = 27;
int: red_1 = 12;

var int: red_2;
var int: black_1;
var int: black_2;

constraint red_1 + red_2 = 26;
constraint black_1 + black_2 = 26;

constraint red_1 + black_1 = deck_1;
constraint red_2 + black_2 = deck_2;

solve satisfy;
```

Variables: Since we know that the first deck contains 12 red cards, we only need to instantiate three variables.

- `red_2`: the red cards of the second deck
- `black_1`: the black cards of the first deck
- `black_2`: the black cards of the second deck

Constraints: Through the first constraint it is specified that the sum of the red cards contained in the first and the second deck is 26. The same applies to the second constraint, which refers to the black cards. The third and fourth constraints specify the amount of red and black cards contained in the first and second decks, respectively.

Solution: Using the line `solve satisfy` it is possible to solve the problem finding the correct value for every variable.

```
red_2 = 14;
black_1 = 13;
black_2 = 13;
```

So the the answer to the problem (How many black cards are there in the second deck?) is 13.

3 Exercise 4 - The smallest

Problem: "What is the smallest number (positive integer) of four digits, all of which are even and different from each other, that is a multiple of 11? (a number cannot can begin with the digit 0)"

Code:

```
include "globals.mzn";

array[1..4] of var 0..9: number;

constraint forall(i in 1..4)((number[i] mod 2) == 0);
constraint number[1] != 0;
constraint all_different(number);
constraint ((number[1]*1000 + number[2]*100 + number[3]*10 + number[4]) mod 11) == 0;

solve minimize (number[1]*1000 + number[2]*100 + number[3]*10 + number[4]);
```

Variables: The problem asks to find a number of four digits, so it is used an array of four variables where each can assume a value from 0 to 9.

Constraints: The solution can be found using four different constraints explained in order:

- the module 2 of every digit is 0, in this way it is assured that the digits are even
- the first digit of the number is different from 0
- the digits are all different from each other
- the module 11 of the number is 0, to assure that the solution found is a multiple of 11

Solution: This exercise requires to find the smallest number, so it is used the line `solve minimize`. The solution is the following:

```
number = [2, 0, 4, 6];
```

4 Exercise 5 - A magic triangle

Problem: "The slots of the triangle in the Figure 1 contain all integer numbers from 1 to 9. Some of the numbers have already been written down. Debora completed the figure, writing the numbers in each slot in such a way that the sum of the numbers that appear on the same side of the triangle is always equal to 20. What number did Debora write in the bottom left vertex slot?"

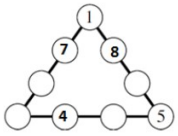

Figure 1: The magic triangle

Code:

```
include "globals.mzn";
array[1..9] of var 1..9: triangle;

constraint triangle[1] == 1;
constraint triangle[2] == 8;
constraint triangle[4] == 5;
constraint triangle[6] == 4;
constraint triangle[9] == 7;

constraint triangle[1] + triangle[2] + triangle[3] + triangle[4] == 20;
constraint triangle[4] + triangle[5] + triangle[6] + triangle[7] == 20;
constraint triangle[7] + triangle[8] + triangle[9] + triangle[1] == 20;

constraint all_different(triangle);

solve satisfy;
```

Variables: To solve this problem, a single array of 9 variables is needed. In this way each variable matches one of the slot in the triangle, starting from the top vertex of the triangle and going around clockwise. An array is preferred over individual variable `var int` since it makes possible to take advantage of a built-in constraint, described in the next section.

Constraints: In this exercise we use three different set of constraints. The constraints from line 4 to 8 manage the slots that have already a number assigned, obtained from Figure 1. The three constraints from line 10 to 12 manages secure the sum of each side of the triangle to be equal to 20 and the last constraint at line 14 makes every element of the array `traingle` different from each other.

Solution: The solution provided is the following:

```
triangle = [1, 8, 6, 5, 2, 4, 9, 3, 7];
```

making the number written by Debora in the bottom left vertex slot equal to 9.

5 Exercise 6 - A maximum triangle

Problem: "Debora, after crossing out all nine numbers (see previous question), issues a challenge to Alexander: - Write the nine numbers, one in each box, so that the sum of the three sums of the numbers that appear on the same side of the triangle is the maximum possible. How much is that sum worth?"

Code:

```
include "globals.mzn";
array[1..9] of var 1..9: triangle;
var int: sum1;
var int: sum2;
var int: sum3;
var int: sum;

constraint triangle[1] + triangle[2] + triangle[3] + triangle[4] == sum1;
constraint triangle[4] + triangle[5] + triangle[6] + triangle[7] == sum2;
constraint triangle[7] + triangle[8] + triangle[9] + triangle[1] == sum3;

constraint all_different(triangle);

constraint sum = sum1 + sum2 + sum3;

solve maximize sum;
```

Variables: Like in the previous exercise, the array of 9 integer variables is useful. In addition to that, the use of three other variables `sum1, sum2, sum3` can be beneficial to find the maximum sum of each side of the triangle.

Constraints: Here, three different type of constraints are used:

- from line 8 to 10: it is specified that the sum of the numbers in the triangle slots for each side has to be equal to the variable sum1, sum2, sum3 respectively
- line 12: `all_different` constrains every element of the array to be different from each other
- line 14: cumulative sum of all sides of the triangle is set to be equal to the variable `sum`, used to maximize the overall total

Solution: Since the goal of the exercise is to maximize the total of the three independent sums of the triangle sides, it is used the `solve maximize`. The solution is the following:

```
1     triangle = [9, 6, 5, 8, 4, 3, 7, 2, 1];
2     sum1 = 28;
3     sum2 = 22;
4     sum3 = 19;
5     sum = 69;
```

In this case it can be noticed that the maximum value obtainable is 69.

6 Exercise 8 - Matteo's age

Problem: "Matteo was born on January 1, 2000. He turned 14 in 2014, and the sum of the digits for that year $(2 + 0 + 1 + 4)$ equals 7, half his his age. In which year, however, will the sum of the digits equal one-fifth of Matteo's age?"

Code:

```
1  array[1..4] of var 0..9: year;
2  var int: current_year;
3  var int: sum;
4  var int: age;
5
6  constraint current_year = year[1]*1000 + year[2]*100 + year[3]*10 + year[4];
7  constraint age == current_year - 2000;
8  constraint sum = sum(year) /\ sum = age / 5;
9
10 solve maximize current_year;
```

Variables: In this exercise three variables and one array of four variables are deployed. The array contains the year to be found digit by digit while `current_year` is the year in integer form. The variable `age` is Matteo's age and `sum` correspond to the sum of the `current_year`'digit that has to be equal to one-fifth of Matteo's age.

Constraints: Three constraints are used to solve this problem, explained as follows:

- line 6: constraint used to set the `current_year` variable, in this way it is also assured that it has only four digits.
- line 7: the age is calculated as the `current_year` from which it is subtracted the year of birth of Matteo.
- line 8: through this constraint, all the varibles are put into relation; `sum` is equal both to the sum of the digits of `year` and to one-fifth of Matteo's age.

Solution: Using `solve maximize current_year` it is possible to find both years requested by the exercise:

```
year = [2, 0, 2, 0];
current_year = 2020;
sum = 4;
age = 20;
```

```
year = [2, 0, 6, 5];
current_year = 2065;
sum = 13;
age = 65;
```

7 Exercise 9 - Four digits for a date

Problem: "19.09.2021 (September 19, 2021) is written using four digits, each employed twice. What will be the next date (written with the same format as the previous one) that has the same property of being written with four digits, each employed twice?"

Code:

```
1  include "globals.mzn";
2  array [1..8] of var 0..9: date;
3  var int: date_n;
4  int: date_prev = 2021*365 + 9*31 + 19;
5
6  constraint date[1] <= 3;
7  constraint date[1] == 3 -> (date[2] == 0 \/ date[2] == 1);
8
9  constraint date[3] <= 1;
10 constraint date[3] == 1 -> (date[4] == 0 \/ date[4] == 1 \/ date[4] == 2);
11
12 constraint date[5] >= 2;
13 constraint (date[5] == 2 /\ date[6] == 0) -> (date[7] >= 2);
14 constraint (date[5] == 2 /\ date[6] == 0 /\ date[7] == 2) -> date[8] > 1;
15
16 constraint forall(i in 1..8)(count(date, date[i], 2));
17
18 constraint date_n = (date[5]*1000 + date[6]*100 + date[7]*10 + date[8])*365 +
19                     (date[3]*10 + date[4])*30 + (date[1]*10 + date[2]);
20
21 constraint date_n > date_prev;
22
23 solve minimize date_n;
```

Variables: To structure an appropriate solution, the following variables and parameter are needed:

- `date` is an array of eight variable that can assume value from 0 to 9; this is the structure that will store the unknown date in the format DD/MM/YYYY
- `date_n` is the variable that contains the unique value corresponding to the date to be found
- `date_prev` is a parameter that contains the unique value corresponding to the date 19/09/2021; the calculation is executed as follows:

$$year * 365 + month * 31 + day$$

Constraints:

- line 6 - 7: days constraints - the digits storing the value of the day can take as a maximum number 31 so `date[1]` must be at most 3; moreover, if `date[1]` takes the value of 3, `date[2]` must necessarily be either 0 or 1, since there are only 30 and 31
- line 9 - 10: month constraints - the digits storing the value of the month can take as a maximum number 12 so `date[3]` must be at most 1; moreover, if `date[3]` takes the value of 1, `date[4]` must necessarily be either 1 or 2

- line 11 - 14: year constraints - the digits that contains the values of the year are constraint to be greater than 2021, as a date after 19/09/2021 is requested
- line 16: checks that the date has 4 digits in it, each employed twice
- line 18: calculate the univocal value, through the formula described above, corresponding to the date to be found
- line 21: checks that the univocal number found, `date_n`, is greater than the one given, `date_prev`; in this way it is assured that the date founded comes after the given one

Solution: Since the objective of the exercise is to find the next date, the variable `date_n` is minimized. The solution is the following:

```
date = [1, 3, 0, 1, 2, 0, 2, 3];
date_n = 738438;
```

The resulting date that respect all the requirements is 13/01/2023.

8 Exercise 11 - The ninth

Problem: "Deleting the digit 0 from the number 405 results in 45, which is its ninth (and still divisible by 9). What is the smallest (positive integer) four-digit number such that deleting a 0 results in its ninth?"

Code:

```
array[1..4] of var 0..9: number;
var int: n;

constraint number[1] != 0;
constraint (number[2] == 0 /\ number[3] != 0 /\ number[4] != 0) \/
           (number[2] != 0 /\ number[3] == 0 /\ number[4] != 0) \/
           (number[2] != 0 /\ number[3] != 0 /\ number[4] == 0);

constraint n = number[1]*1000 + number[2]*100 + number[3]*10 + number[4];

constraint (number[2] == 0 /\ ((number[1]*100 + number[3]*10 + number[4]) == n/9 )) \/
           (number[3] == 0 /\ ((number[1]*100 + number[2]*10 + number[4]) == n/9 )) \/
           (number[4] == 0 /\ ((number[1]*100 + number[2]*10 + number[3]) == n/9 ));

solve minimize n;
```

Variables: The problem asks to find a four digit number, so an array of four variables would be very useful. In addition, there is the need of an integer variable, used to store the number in the array in integer form.

Constraints:

- the first constraint establish that the left-most digit of the number is not null
- the second constraint assures that in the number searched is present one and only one zero
- the third constraint makes the number contained in the array in an integer form contained in the variable `n`
- the fourth constraint assures that one of the digits is 0 and simultaneously that by removing it, the new number results in the ninth of the starting number

Solution: The problem asks to find the smallest number, hence minimizing the variable `n`. The solution is the following:

```
number = [2, 0, 2, 5];
n = 2025;
```

9 Exercise 12 - Magic!

Problem: "Carla takes a number (positive integer) of two digits, multiplies it by 4 and then subtracts 3 from the result obtained. Magic: the final number obtained by Carla is written with the same digits as the starting number, but in reverse order. What was the starting number?"

Code:

```
1  var 0..9: n1;
2  var 0..9: n2;
3  var int: n;
4  var int: f;
5
6  constraint n = n1*10 + n2;
7  constraint f = n2*10 + n1;
8  constraint f = ((n*4) - 3);
9
10 solve satisfy;
```

Variables: Using four variables is useful: `n1` and `n2` are the single digits that make up the number, while `n` is the starting number and `f` is the final one derived from the operations.

Constraints:

- line 6: establish the order of the digits of the starting number
- line 7: establish the order of the final number as reversed
- line 8: this is the key constraint since it is an equivalence between the final number and the starting number to which are applied Carla's multiplication and division

Solution: Solving this exercise produce the following result:

```
1 n1 = 1;
2 n2 = 6;
3 n = 16;
4 f = 61;
```

Therefore the starting number is 16.

10 Exercise 13 - Carla's boxes

Problem: "Carla has at her disposal the two boxes in Figure 2 (which are cubes: the first one has a side of 10 cm, the second one 20 cm). She fills the small one with water several times, to the brim, and then pours it into the second box, without losing a single drop. How many decants will Carla be able to do, at most, to fill the second box?"

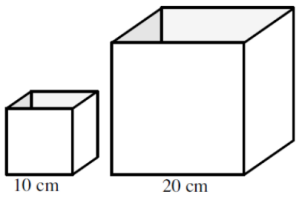

Figure 2: Carla's boxes

Code:

```
int: small_box = 10;
int: big_box = 20;
var int: n_pours;

constraint n_pours = pow(big_box, 3) / pow(small_box, 3);

solve satisfy;
```

Variables: For this exercise only one variable is needed, `n_pour` that is the number of pours.

Constraints: The only constraint used for this problem is the division between the volume of the big box and the volume of the little box. In this way it is easy to find the number of times the little box is contained into the big one.

$$n\_pours = \frac{V_{big\_box}}{V_{small\_box}}$$

Solution: The numbers of decants needed to fill the second box are 8.

```
n_pours = 8;
```

11 Exercise 14 - The wooden solid

Problem: "If you cut a small part of a wooden cube in the area around one of its vertices, you will get a solid with seven faces. How many faces will you get, at most, if in the same way you then cut a small part of the solid with seven faces, in the areas around each of its vertices?"

Code:

```
int: n_cuts = 7;
int: vertex_init = 10;

array[1..vertex_init] of var int: face_cuts;
array[1..vertex_init] of var int: vertex_cuts;

constraint forall(i in 1..vertex_init)(if i==1
                                       then face_cuts[i] = n_cuts + 1
                                       else face_cuts[i] = face_cuts[i-1] + 1
                                       endif);

constraint forall(i in 1..vertex_init)(if i==1
                                       then vertex_cuts[i] = vertex_init + 2
                                       else vertex_cuts[i] = vertex_cuts[i-1] + 2
                                       endif);

solve satisfy;
```

Variables:

- `face_cuts` an array of variables that has in each element the updated number of total faces after each cut
- `vertex_cuts` an array of variables that has in each element the updated number of total vertexes after each cut

For this specific exercise it is also useful to explain the parameters:

- `n_cuts` total number of faces of the solid at the starting point
- `vertex_init` number of vertex of the solid, so the total number of cuts that has to be performed

Constraints: Only two constraint are used to find the solution, both are `forall` cycles that counts how many new cuts and vertexes for each cut (each cycle) there are. The cycle keeps going for `vertex_init` times, that is the total number of vertices to be cut.

Solution: Here it can be seen the correspondence between the number of vertices and faces:

```
face_cuts = [8, 9, 10, 11, 12, 13, 14, 15, 16, 17];
vertex_cuts = [12, 14, 16, 18, 20, 22, 24, 26, 28, 30];
```

The final number of faces of the new solid is 17.

12 Exercise 17 - With the ten digits

Problem: "A number N, positive integer, is such that all ten digits from 0 to 9 are used to write its third power, N3, and its fourth power, N4. What is the value of N?"

Code:

```
include "globals.mzn";

var int: n;
var int: n3;
var int: n4;

array[1..9] of var 0..9: n3_array;
array[1..9] of var 0..9: n4_array;

constraint n3 = sum(i in 1..9)(10^(9 - i) * n3_array[i]);
constraint n4 = sum(i in 1..9)(10^(9 - i) * n4_array[i]);

constraint n3 = pow(n, 3);
constraint n4 = pow(n, 4);

constraint forall(i in 1..9)(exists(j in 1..9)(n3_array[j] = i \/ n4_array[j] = i));

solve satisfy;
```

Variables:

- `n` is the variable that contains the number to find
- `n3` is the variable that contains the third power of `n`
- `n4` is the variable that contains the fourth power of `n`
- `n3_array` is the array of four variables that stores every digit of the third power of `n`
- `n4_array` is the array of four variables that stores every digit of the fourth power of `n`

Constraints:

- line 10 - 11 these constraints convert the numbers stored in the array in their respective integer forms
- line 13 - 14 these constraints set respectively the values of `n3` and `n4` equal to `n` raised to the power of 3 and 4
- line 16 this constraint assigns all 10 digits (0 to 9) to `n3_array` and `n4_array` in such a way that they are all only used once to write `n3` and `n4`

Solution: The output of the exercise it the following:

```
n = 18;
n3 = 5832;
n4 = 104976;
n3_array = [0, 0, 0, 0, 0, 5, 8, 3, 2];
n4_array = [0, 0, 0, 1, 0, 4, 9, 7, 6];
```

The value of N is 18.