EDAN01
Introduction to Constraint Programming
Lab Instruction

Authors: Krzysztof Kuchcinski
Krzysztof.Kuchcinski@cs.lth.se

Course responsible: Krzysztof Kuchcinski (krzysztof.kuchcinski@cs.lth.se)
Lab assistants: Mehmet Ali Arslan (mehmet_ali.arslan@cs.lth.se)
Gustav Cedersjö (gustav.cedersjo@cs.lth.se)

December 16, 2015

1

Lab assignments

To pass the labs you need to solve five lab assignments specified in chapter 4 in this instruction. There are also simple introductory assignments that you may try to learn constraint programming in JaCoP and minizinc. The following constraints apply.

- For each assignment there is a deadline. For details look at the assignments.
- At the deadline you need to report your results together with the program to the lab assistant.
- All programs needs also to be send to edan01@cs.lth.se with subject "student's id" and {Uppgift 1 | Uppgift 2 | Uppgift 3 | Uppgift 4 | Uppgift 5 }.
- At least on assignment, excluding Search assignment from section 4.3, need to be done using JaCoP in Java and at least one with minizinc.
- The programs must not be hard-coded for given parameters. You need to program solutions that they use data included in the Java program or, in case of minizinc, as separate dzn file.

2

Introduction to JaCoP nad Minizinc

The Finite domain solver can be used when problem variables can be modeled as integers. Many problems are of this nature and therefore most of your work will concentrate on this part. For the labs you will use JaCoP finite domain library (Java Constraint Programming library). A separate document describes this library.

2.1 Introductory Example

Consider classical SEND + MORE = MONEY example. It is a simple arithmetic logic puzzle, where digits are represented by the letters and different letters represent different digits. The goal is to find assignment of digits to letters that SEND+MORE=MONEY, i.e. addition of numbers represented by SEND and MORE is equal MONEY. Moreover, S and M cannot be equal zero.

2.1.1 Java solution

```
import org.jacop.constraints.*;
import org.jacop.core.*;
import org.jacop.search.*;

public class SendMoreMoney  {

    public static void main(String[] args) {
        long T1, T2, T;
        T1 = System.currentTimeMillis();

        send();

        T2 = System.currentTimeMillis();
        T = T2 - T1;
        System.out.println("\n\t*** Execution time = " + T + " ms");
    }

    static void send() {
```

```
        Store store = new Store();

        IntVar s = new IntVar(store, "S", 0, 9);
        IntVar e = new IntVar(store, "E", 0, 9);
        IntVar n = new IntVar(store, "N", 0, 9);
        IntVar d = new IntVar(store, "D", 0, 9);
        IntVar m = new IntVar(store, "M", 0, 9);
        IntVar o = new IntVar(store, "O", 0, 9);
        IntVar r = new IntVar(store, "R", 0, 9);
        IntVar y = new IntVar(store, "Y", 0, 9);

        IntVar digits[] = { s, e, n, d, m, o, r, y };

        store.impose(new Alldiff(digits));

        /**
         * Constraint for equation
         * SEND = 1000 * S + 100 * E + N * 10 + D * 1
         * MORE = 1000 * M + 100 * O + R * 10 + E * 1
         * MONEY = 10000 * M + 1000 * O + 100 * N + E * 10 + Y * 1
         */
        store.impose(new LinearInt(store,
              new IntVar[] {s, e, n, d, m, o, r, e, m, o, n, e, y},
              new int[] {1000, 100, 10, 1, 1000, 100, 10, 1, -10000, -1000, -100, -10, -1},
              "==", 0));

        store.impose(new XneqC(s, 0));
        store.impose(new XneqC(m, 0));

        System.out.println("Number of variables: "+ store.size() +
                           "\nNumber of constraints: " + store.numberConstraints());

        Search<IntVar> search = new DepthFirstSearch<IntVar>();
        SelectChoicePoint<IntVar> select = new SimpleSelect<IntVar>(digits,
                                                    new SmallestDomain<IntVar>(),
                                                    new IndomainMin<IntVar>());
        search.setSolutionListener(new PrintOutListener<IntVar>());
        search.getSolutionListener().searchAll(true);

        boolean Result = search.labeling(store, select);

        if (Result) {
            System.out.println("\n*** Yes");
            System.out.println("Solution : "+ java.util.Arrays.asList(digits));
        }
        else System.out.println("\n*** No");
    }
}
```

JaCoP finds the solution.

```
Number of variables: 8
Number of constraints: 4


Solution : [S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2]

Depth First Search DFS0
[S = 9, M = 1, O = 0, E = 5, D = 7, N = 6, R = 8, Y = 2]

Solution : [S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2]
Nodes : 6
Decisions : 3
Wrong Decisions : 3
Backtracks : 3
Max Depth : 3


*** Yes
Solution : [S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2]

        *** Execution time = 21 ms
```

2.1.2 MiniZinc solution

The same examples can also be specified in minizinc using the following program.

```
include "globals.mzn";

var 0..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 0..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
array[1..8] of var int : fd = [S,E,N,D,M,O,R,Y];

constraint
  all_different(fd) /\
            1000*S + 100*E + 10*N + D  +
            1000*M + 100*O + 10*R + E  =
  10000*M + 1000*O + 100*N + 10*E + Y
  /\ S > 0 /\ M > 0;

solve satisfy;

output[show(fd)];
```

After compilation to flatzinc and execution the program finds the same answer.

```
S = 9;
```

```
E = 5;
N = 6;
D = 7;
M = 1;
O = 0;
R = 8;
Y = 2;
----------
%% Model variables : 8
%% Model constraints : 3

%% Search CPU time : 4ms
%% Search nodes : 3
%% Propagations : 19
%% Search decisions : 2
%% Wrong search decisions : 1
%% Search backtracks : 0
%% Max search depth : 2
%% Number solutions : 1

%% Total CPU time : 82ms
```

3

Introductory exercises- not obligatory

You may use the exercises of this section to learn the JaCoP solver and/or minizinc language. The examples are rather simple and the efforts are on modeling and solving.

3.1 Lab assignment – Good Burger[1]

As the owner of a fast food restaurant with declining sales, your customers are looking for something new and exciting on the menu. Your market research indicates that they want a burger that is loaded with everything as long as it meets certain health requirements. Money is no object to them.

The ingredient list in the table shows what is available to include on the burger. You must include at least one of each item and no more than five of each item. You must use whole items (for example, no half servings of cheese). The final burger must contain less than 3000 mg of sodium, less than 150 grams of fat, and less than 3000 calories.

To maintain certain taste quality standards you'll need to keep the servings of ketchup and lettuce the same. Also, you'll need to keep the servings of pickles and tomatoes the same.

Question: What is the most expensive burger you can make? How many different products must be used?

Item	Sodium(mg)	Fat(g)	Calories	Item cost($)
Beef Patty	50	17	220	$0.25
Bun	330	9	260	$0.15
Cheese	310	6	70	$0.10
Onions	1	2	10	$0.09
Pickles	260	0	5	$0.03
Lettuce	3	0	4	$0.04
Ketchup	160	0	20	$0.02
Tomato	3	0	9	$0.04

Table 3.1: Table of ingredients for burgers.

[1] Based on `http://puzzlor.com/2014-08_GoodBurger.html`

3.2 Lab assignment – Chandelier Balancing[2]

Problem:

Constructing a chandelier can be a tricky undertaking because the slightest imperfection will unbalance the chandelier and cause it to be skewed.

Figure 1 shows a chandelier constructed from arms, wires and triangles that hold weights. In order to perfectly balance the chandelier, weights must be placed into the triangles. There are nine weights as follows: 1, 2, 3, 4, 5, 6, 7, 8, and 9 kg. Each triangle can only hold one weight. Assume the weight of the arms, wires and triangles are negligible.

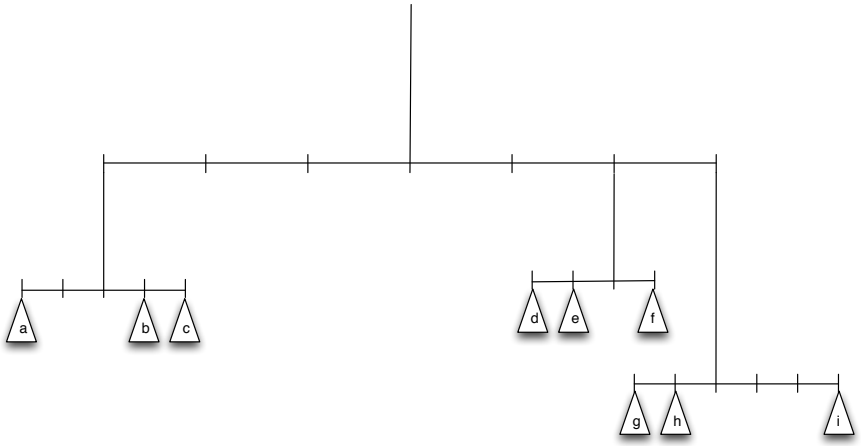

Figure 3.1: Balancing act: Where should the weights be placed?

Question: Where should the weights be placed in order to perfectly balance the chandelier?

[2]John Toczek, https://www.informs.org/ORMS-Today/Public-Articles/February-Volume-40-Number-1/THE-PUZZLOR

4

Lab Assignments

You have to use JaCoP library together with Java or minizinc.

4.1 Lab assignment – Pizza[1]

The problem arises in the University College Cork student dorms. The students select from the meny `n` pizzas they want to order and then minimize the price for the whole order by using the vouchers for acquiring discounts in purchasing pizzas.

A voucher is a pair of numbers e.g. (2,4), which means if you pay for 2 pizzas then you can obtain for free up to 4 pizzas as long as they each cost no more than the cheapest of the 2 pizzas you paid for. Similarly a voucher (3,2) means that if you pay for 3 pizzas you can get up to 2 pizzas for free as long as they each cost no more than the cheapest of the 3 pizzas you paid for.

The goal is to obtain `n` pizzas for the least possible cost. Note that not all vouchers need to be used, and a voucher does not need to be totally used.

Table 4.1 specifies three possible inputs that you should use for the problem. It also contains for your convenience the expect cost of the solution. There following constants are used.

- n - number of pizzas to order,
- price – list of n pizza prices,
- m – number of vouchers, and
- two lists of size m named `buy` and `free`, which specify the vouchers. For i in 1..m, one can get free[i] free pizzas when buying buy[i] pizzas.

The program in Java or minizinc should work on different data, i.e. there is only one program that can be run on different input data.

[1] LP/CP Programming Contest 2015, `http://picat-lang.org/lp_cp_pc/Pizza.html`

Java input	Minizinc input	Output Cost

`int n = 4;`
`int[] price = {10,5,20,15};`
`int m = 2;`
`int[] buy = {1,2};`
`int[] free = {1,1};`
	`n = 4;`
	`price = [10,5,20,15];`
	`m = 2;`
	`buy = [1,2];`
	`free = [1,1];`
		35

`int n = 4;`
`int[] price = {10,15,20,15};`
`int m = 7;`
`int[] buy = {1,2,2,8,3,1,4};`
`int[] free = {1,1,2,9,1,0,1};`
	`n = 4;`
	`price = [10,15,20,15];`
	`m = 7;`
	`buy = [1,2,2,8,3,1,4];`
	`free = [1,1,2,9,1,0,1];`
		35

`int n = 10;`
`int[] price = {70,10,60,60,30,`
`100,60,40,60,20};`
`int m = 4;`
`int[] buy = {1,2,1,1};`
`int[] free = {1,1,1,0};`
	`n = 10;`
	`price = [70,10,60,60,30,`
	`100,60,40,60,20];`
	`m = 4;`
	`buy = [1,2,1,1];`
	`free = [1,1,1,0];`
		340

Table 4.1: Inputs and expected cost for the solutions for the pizza example.

4.2 Lab assignment – Logistics[2]

John is a truck driver. He needs to deliver a truckload of packages to different destination cities. In each city other truck drivers can help transport part of the packages with their trucks. So nobody is required to solve the traveling salesman problem. Also, no driver is required to return to his starting city. Nevertheless, John has to pay for the distances the trucks travel, and he wants to find routes for himself and the helpers such that the total travel distance is minimized.

The problem can be formulated as a graph problem as follows: Given an undirected weighted graph, a starting vertex, and a set of destination vertices, find a subgraph of the graph that has the minimum cost and covers the starting vertex and all of the destination vertices.

For example consider figure 4.1, for graph (a) shown below, assume the starting vertex is 1, and the set of destination vertices is 5,6, then graph (b) shows a minimum covering tree. The total distance in this example is $1 + 6 + 4 = 11$

Table 4.2 specifies three possible inputs that you should use for the problem. It also contains for your convenience the expect cost of the solution. There are defined the following constants.

An input specifies the size of the graph (`graph_size`), the starting vertex (`start`), the number of destinations (`n_dests`), an array of destination vertices (`dest`), the number of edges (`n_edges`), and three arrays that define the edge relation (`from`, `to`, and `cost`).

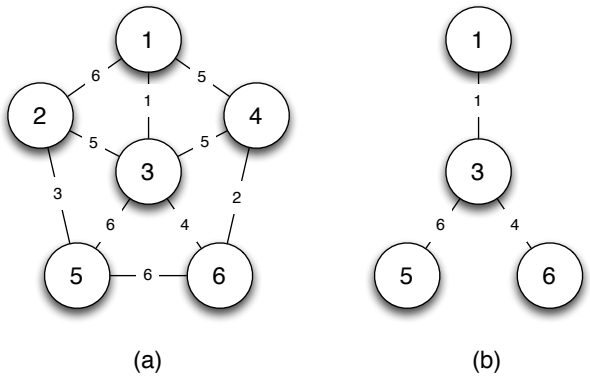

Figure 4.1: An example graph for assignment 4.2

[2]LP/CP Programming Contest 2015, `http://picat-lang.org/lp_cp_pc/Logistics.html`

Java input	Minizinc input	Output cost

`int graph_size = 6;`
`int start = 1;`
`int n_dests = 1;`
`int[] dest = {6};`
`int n_edges = 7;`
`int[] from = {1,1,2,2,3,4,4};`
`int[] to = {2,3,3,4,5,5,6};`
`int[] cost = {4,2,5,10,3,4,11};`
	`graph_size = 6;`
	`start = 1;`
	`n_dests = 1;`
	`dest = [6];`
	`n_edges = 7;`
	`from = [1,1,2,2,3,4,4];`
	`to = [2,3,3,4,5,5,6];`
	`cost = [4,2,5,10,3,4,11];`
		20

`int graph_size = 6;`
`int start = 1;`
`int n_dests = 2;`
`int[] dest = {5,6};`
`int n_edges = 7;`
`int[] from = {1,1,2, 2,3,4, 4};`
`int[] to = {2,3,3, 4,5,5, 6};`
`int[] cost = {4,2,5,10,3,4,11};`
	`graph_size = 6;`
	`start = 1;`
	`n_dests = 2;`
	`dest = [5,6];`
	`n_edges = 7;`
	`from = [1,1,2, 2,3,4, 4];`
	`to =  [2,3,3, 4,5,5, 6];`
	`cost = [4,2,5,10,3,4,11];`
		20

`int graph_size = 6;`
`int start = 1;`
`int n_dests = 2;`
`int[] dest = {5,6};`
`int n_edges = 9;`
`int[] from = {1,1,1,2,2,3,3,3,4};`
`int[] to = {2,3,4,3,5,4,5,6,6};`
`int[] cost = {6,1,5,5,3,5,6,4,2};`
	`graph_size = 6;`
	`start = 1;`
	`n_dests = 2;`
	`dest = [5,6];`
	`n_edges = 9;`
	`from = [1,1,1,2,2,3,3,3,4];`
	`to = [2,3,4,3,5,4,5,6,6];`
	`cost = [6,1,5,5,3,5,6,4,2];`
		11

Table 4.2: The input data and the expected output cost for assignment 4.2

4.3 Lab assignment– Depth First Search

File `SimpleDFS.java` contains a simple but fully functional depth first search (DFS). It implements basic functionality of DFS and Branch-and-Bound for minimization. The program is very simple since it makes it possible to select variables based on the order defined by the variable vector and it always assigns a minimal value to each variable.

You can try the search on the included example, Golomb.java. It is minimization example and it is described, for example in `http://mathworld.wolfram.com/GolombRuler.html`.

In this assignment you have to change the behavior of this search. You have to implement a split search that selects a variable based on input order, as it is done in SimpleDFS but then narrows the domain of the variable without assigning a single value to the selected variable.

Two search strategies has to be implemented.

1. The first search makes the following choice point on selected variable x.

   - first choice $x \leq c$, and if this fails
   - second choice $x > c$, that is negated $x \leq c$.

   where c is a middle point of the domain interval, that is $c = \frac{x.max()+x.min()}{2}$.

2. The second search makes a different choice point on selected variable x.

   - first choice $x \geq c$, and if this fails
   - second choice $x < c$, that is negated $x \geq c$.

   where c is a middle point of the domain interval, that is $c = \frac{x.max()+x.min()}{2}$.

Finally, experiment with different variable selection methods and select the best method for this example.

Test your solution on the included example and report the following statistics for the search (requires some code in the search).

- total number of search nodes, and
- number of wrong decisions

Report your results in table 4.3.

Hint:

Program `SimpleDFS` has variable `trace`. Setting this variable to `true` forces the search to output current search variables every time the search enters a new search node. It might help you to find errors in your implementation.

	Total # Nodes	Wrong Decisions
SimpleDFS		
SplitSearch 1		
SplitSearch 2		
Variable selection method:		

Table 4.3: Results for different kinds of search strategies

4.4 Lab assignment – Auto Regression Filter

Problem:

The data-flow graph for the auto regression filter is depicted on Figure 4.2. It consists of 16 multiplications and 12 additions. These operations need to be scheduled on multipliers and adders. Write a program which will

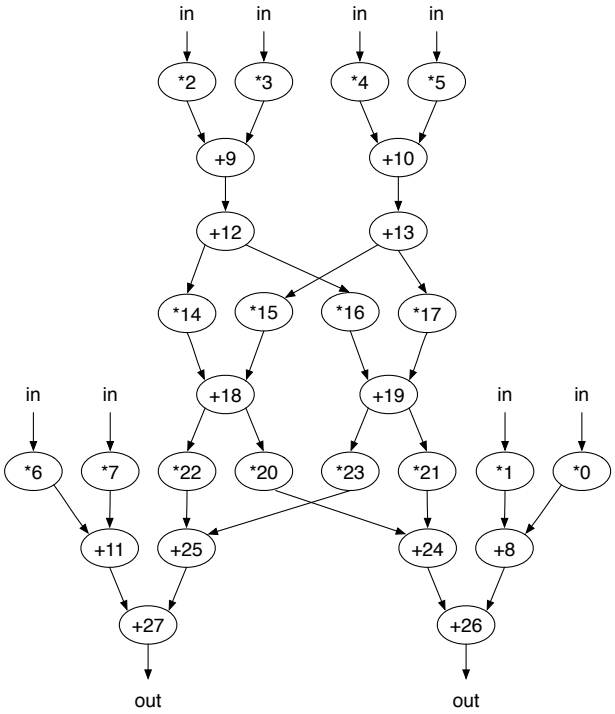

Figure 4.2: The operation graph representation of the auto regression filter.

optimize the schedule length for different amount of resources as specified in table 4.4. Fill the table with results obtained from your program, Assume that multiplication takes two clock cycles and addition only one, but write your program in such a way that you can easily specify otherwise. Make your program **data independent**, so if new operation or new operation type is added the program does not have to be changed but only the database of facts. Note, that this means that the graph can change.

Hint: An efficient model should use `Cumulative` and `Diff2` constraints. You may need to use special labeling.

Configuration	Operation assignment	Clock cycles	runtime [s]	optimal [YES/NO]
1 adder, 1 multiplier	adder 1: mul 1:			
1 adder, 2 multiplier	adder 1: mul 1: mul 2:			
1 adder, 3 multiplier	adder 1: mul 1: mul 2: mul 3:			
2 adder, 2 multiplier	adder 1: adder 2: mul 1: mul 2:			
2 adder, 3 multiplier	adder 1: adder 2: mul 1: mul 2: mul 3:			
2 adder, 4 multiplier	adder 1: adder 2: mul 1: mul 2: mul 3: mul 4:			

Table 4.4: Experimental results

4.5 Lab assignment – Urban Planning[3]

Urban planning requires careful placement and distribution of commercial and residential lots. Too many commercial lots in one area leave no room for residential shoppers. Conversely, too many residential lots in one area leave no room for shops or restaurants.

Figure 4.3: An example of residential area with commercial lots.

The 5x5 grid in depicted in Figure 4.3 shows a sample configuration of residential ( ) and commercial lots ( ). Your job is to reorder the 12 residential lots and 13 commercial lots to maximize the quality of the layout. The quality of the layout is determined by a points system. Points are awarded as follows:

- Any column or row that has 5 Residential lots = +5 points
- Any column or row that has 4 Residential lots = +4 points
- Any column or row that has 3 Residential lots = +3 points
- Any column or row that has 5 Commercial lots = -5 points
- Any column or row that has 4 Commercial lots = -4 points
- Any column or row that has 3 Commercial lots = -3 points

For example, the layout displayed in Figure 1 has a total of 9 points: Points for each column, from left to right = -3, -5, +3, +4, +3 Points for each row, from top to bottom = +3, +3, +3, +3, -5.

Solve this problem for input data specified in Table 4.5.

Question: What is the maximum number of points you can achieve for the layout? Give the layout for this case.

Hint: The problem contains many equivalent symmetrical solution and to make it efficient you need to add symmetry breaking constraints.

[3]Based on `http://puzzlor.com/2013-08_UrbanPlanning.html`

Java input	Minizinc input	Output cost

`int n = 5;`
`int n_commercial = 13;`
`int n_residential = 12;`
`int[] point_distribution =`
`{-5, -4, -3, 3, 4, 5};`
	`n = 5;`
	`n_commercial = 13;`
	`n_residential = 12;`
	`point_distribution = array1d(0..n,`
	`[-5, -4, -3, 3, 4, 5]);`
		14

`int n = 5;`
`int n_commercial = 7;`
`int n_residential = 18;`
`int[] point_distribution =`
`{-5, -4, -3, 3, 4, 5};`
	`n = 5;`
	`n_commercial = 7;`
	`n_residential = 18;`
	`point_distribution = array1d(0..n,`
	`[-5, -4, -3, 3, 4, 5]);`
		36

`int n = 7;`
`int n_commercial = 20;`
`int n_residential = 29;`
`int[] point_distribution =`
`{-7, -6, -5, -4, 4, 5, 6, 7};`
	`n = 7;`
	`n_commercial = 20;`
	`n_residential = 29;`
	`point_distribution = array1d(0..n,`
	`[-7, -6, -5, -4, 4, 5, 6, 7]);`
		58

Table 4.5: Input data for urban planning example.