

EDAN01  
Introduction to Constraint Programming  
Lab Instruction

**Author:** Krzysztof Kuchcinski  
Krzysztof.Kuchcinski@cs.lth.se

**Course responsible:** Krzysztof Kuchcinski (krzysztof.kuchcinski@cs.lth.se)

**Lab assistants:** Alfred Åkesson (alfred.akesson@cs.lth.se)  
Noric Couderc (noric.couderc@cs.lth.se)  
Jonas Skeppstedt (jonas.skeppstedt@cs.lth.se)

October 22, 2018

# 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 one 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.

Examples of java and dzn files can be found in course directory at /usr/local/cs/EDAN01/data/.

### IMPORTANT

Use `initcs` command first at lab computers to load environment variables for this course.

## 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  $\text{SEND} + \text{MORE} = \text{MONEY}$  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  $\text{SEND} + \text{MORE} = \text{MONEY}$ , 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

#### Java code

```
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(
        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,
        null,
        new IndomainMin<IntVar>());
    search.setSolutionListener(new PrintOutListener<IntVar>());

    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.

#### 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, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2]

```

```

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

```

```

Nodes : 3

```

```

Decisions : 2

```

```

Wrong Decisions : 1

```

```

Backtracks : 0

```

```

Max Depth : 2

```

```

*** Yes

```

```

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

```

```

*** Execution time = 25 ms

```

### 2.1.2 MiniZinc solution

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

#### Minizinc code

```

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 with command  
mzn2fzn -Gjacop sendmory.mzn  
and execution with the command  
fzn-jacop -s sendmory.fzn  
the program finds the same answer.

#### Solution

```
S = 9;
E = 5;
N = 6;
D = 7;
M = 1;
O = 0;
R = 8;
Y = 2;
-----
%%mzn-stat: variables=8
%%mzn-stat: propagators=2

%%mzn-stat: initTime=0.055
%%mzn-stat: solveTime=0.005
%%mzn-stat: nodes=3
%%mzn-stat: propagations=21
%%mzn-stat: failers=1
%%mzn-stat; peakDepth=2
%%mzn-stat: solutions=1

%%mzn-stat: time=0.060
```

### 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<sup>1</sup>

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.

---

<sup>1</sup>Based on [http://puzzlor.com/2014-08\\_GoodBurger.html](http://puzzlor.com/2014-08_GoodBurger.html)

### 3.2 Lab assignment – Chandelier Balancing<sup>2</sup>

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.

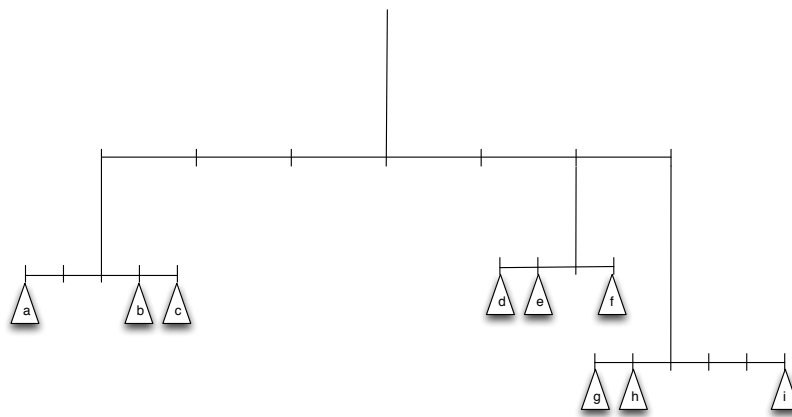


Figure 3.1: Balancing act: Where should the weights be placed?

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

<sup>2</sup>John Toczec, <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 – Photo

**DEADLINE**

**week 46**

A group of  $n$  people wants to take a group photo. Each person can give preferences next to whom he or she wants to be placed on the photo. The problem to be solved is to find a placement that satisfies maximum number of preferences.

You should solve the original problem stated above and the modified problem. In the modified problem, you need to minimize the distances between people on the preference list, i.e., minimize the maximal distance between two persons from the preference list. Compare the solutions.

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.

Java input	Minizinc input	Output Cost
<pre> <b>int</b> n = 9; <b>int</b> n_prefs = 17; <b>int</b>[][] prefs = {{1,3}, {1,5}, {1,8},   {2,5}, {2,9}, {3,4}, {3,5}, {4,1},   {4,5}, {5,6}, {5,1}, {6,1}, {6,9},   {7,3}, {7,8}, {8,9}, {8,7}}; </pre>	<pre> n = 9; n_prefs = 17; prefs = [  1,3   1,5   1,8   2,5   2,9     3,4   3,5   4,1   4,5   5,6   5,1     6,1   6,9   7,3   7,8   8,9   8,7  ]; </pre>	10(4)
<pre> <b>int</b> n = 11; <b>int</b> n_prefs = 20; <b>int</b>[][] prefs = {1,3}, {1,5}, {2,5},   {2,8}, {2,9}, {3,4}, {3,5}, {4,1},   {4,5}, {4,6}, {5,1}, {6,1}, {6,9},   {7,3}, {7,5}, {8,9}, {8,7}, {8,10},   {9, 11}, {10, 11}; </pre>	<pre> n = 11; n_prefs = 20; prefs = [  1,3   1,5   2,5   2,8   2,9     3,4   3,5   4,1   4,5   4,6   5,1     6,1   6,9   7,3   7,5   8,9   8,7     8,10   9, 11   10, 11  ]; </pre>	11(4)
<pre> <b>int</b> n = 15; <b>int</b> n_prefs = 20; <b>int</b>[][] prefs = {1,3}, {1,5}, {2,5},   {2,8}, {2,9}, {3,4}, {3,5}, {4,1},   {4,15}, {4,13}, {5,1}, {6,10}, {6,9},   {7,3}, {7,5}, {8,9}, {8,7}, {8,14},   {9, 13}, {10, 11}; </pre>	<pre> n = 15; n_prefs = 20; prefs = [  1,3   1,5   2,5   2,8   2,9     3,4   3,5   4,1   4,15   4,13   5,1     6,10   6,9   7,3   7,5   8,9   8,7     8,14   9, 13   10, 11  ]; </pre>	13(3)

Table 4.1: Inputs and expected cost for the solutions for the photo example. In parenthesis there is the cost for the second case.

## 4.2 Lab assignment – Logistics<sup>1</sup>

**DEADLINE**

**week 48**

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 expected 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`).

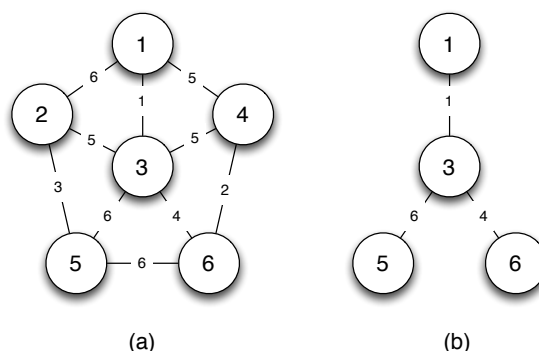


Figure 4.1: An example graph for assignment 4.2

<sup>1</sup>LP/CP Programming Contest 2015, [http://picat-lang.org/lp\\_cp\\_pc/Logistics.html](http://picat-lang.org/lp_cp_pc/Logistics.html)

Java input	Minizinc input	Output cost
<pre> <b>int</b> graph_size = 6; <b>int</b> start = 1; <b>int</b> n_dests = 1; <b>int</b>[] dest = {6}; <b>int</b> n_edges = 7; <b>int</b>[] from = {1,1,2,2,3,4,4}; <b>int</b>[] to = {2,3,3,4,5,5,6}; <b>int</b>[] cost = {4,2,5,10,3,4,11}; </pre>	<pre> 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]; </pre>	20
<pre> <b>int</b> graph_size = 6; <b>int</b> start = 1; <b>int</b> n_dests = 2; <b>int</b>[] dest = {5,6}; <b>int</b> n_edges = 7; <b>int</b>[] from = {1,1,2, 2,3,4, 4}; <b>int</b>[] to = {2,3,3, 4,5,5, 6}; <b>int</b>[] cost = {4,2,5,10,3,4,11}; </pre>	<pre> 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]; </pre>	20
<pre> <b>int</b> graph_size = 6; <b>int</b> start = 1; <b>int</b> n_dests = 2; <b>int</b>[] dest = {5,6}; <b>int</b> n_edges = 9; <b>int</b>[] from = {1,1,1,2,2,3,3,3,4}; <b>int</b>[] to = {2,3,4,3,5,4,5,6,6}; <b>int</b>[] cost = {6,1,5,5,3,5,6,4,2}; </pre>	<pre> 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]; </pre>	11

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

### 4.3 Lab assignment– Depth First Search

#### DEADLINE

week 49

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	16034	7791
SplitSearch 1	22425	11170
SplitSearch 2	697987	348881
Variable selection method:		

Table 4.3: Results for different kinds of search strategies

#### 4.4 Lab assignment – Auto Regression Filter

**DEADLINE**

**week 50**

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

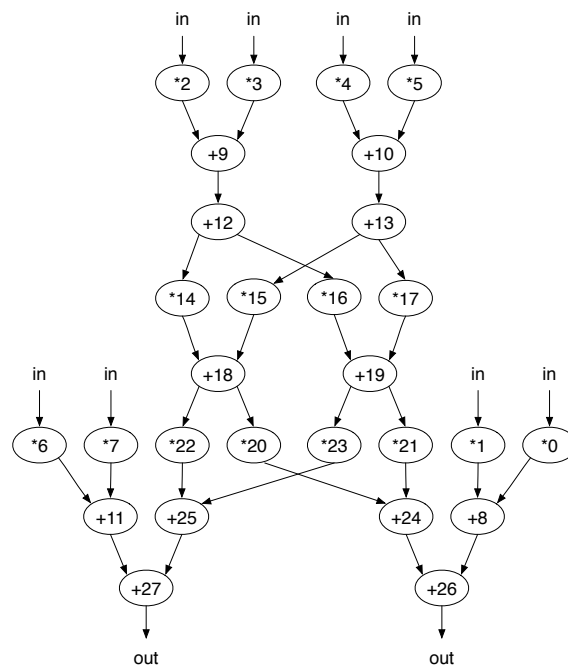


Figure 4.2: The operation graph representation of the auto regression filter.

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.

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

**Hints:**

- An efficient model should use Cumulative and Diff2/Diffn constraints. You may need to use special labeling.
- The example with 1 adder and 3 multipliers may take several minutes to compute.



## 4.5 Lab assignment – Urban Planning<sup>2</sup>

**DEADLINE**

**week 51**

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.

<sup>2</sup>Based on [http://puzzlor.com/2013-08\\_UrbanPlanning.html](http://puzzlor.com/2013-08_UrbanPlanning.html)

Java input	Minizinc input	Output cost
<pre>int n = 5; int n_commercial = 13; int n_residential = 12; int[] point_distribution =     {-5, -4, -3, 3, 4, 5};</pre>	<pre>n = 5; n_commercial = 13; n_residential = 12; point_distribution =     [-5, -4, -3, 3, 4, 5];</pre>	14
<pre>int n = 5; int n_commercial = 7; int n_residential = 18; int[] point_distribution =     {-5, -4, -3, 3, 4, 5};</pre>	<pre>n = 5; n_commercial = 7; n_residential = 18; point_distribution =     [-5, -4, -3, 3, 4, 5];</pre>	36
<pre>int n = 7; int n_commercial = 20; int n_residential = 29; int[] point_distribution =     {-7, -6, -5, -4, 4, 5, 6, 7};</pre>	<pre>n = 7; n_commercial = 20; n_residential = 29; point_distribution =     [-7, -6, -5, -4, 4, 5, 6, 7];</pre>	58

Table 4.5: Input data for urban planning example.

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.