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 assistant: Mehmet Ali Arslan
Mehmet_Ali.Arslan@cs.lth.se

January 9, 2015

1

Lab assignments

To pass the labs you need to solve four lab assignments specified chapters 4 and 5 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 | Search}.
- At least on assignment from chapter 4 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 include in the Java program or, in case of minizinc, as separate dzn file. Examples of dzn files can be found in course directory at `/usr/local/cs/EDAN01/data_dzn/`.

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

    boolean Result = label.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.

```
fd = array1d(1..8, [9, 5, 6, 7, 1, 0, 8, 2]);
----------
==========

%% Model variables : 10
%% Model constraints : 2

%% Search CPU time : 4ms
%% Search nodes : 6
%% Search decisions : 3
%% Wrong search decisions : 3
%% Search backtracks : 3
%% Max search depth : 3
%% Number solutions : 1

%% Total CPU time : 70ms
```

3

Introductory examples- 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.

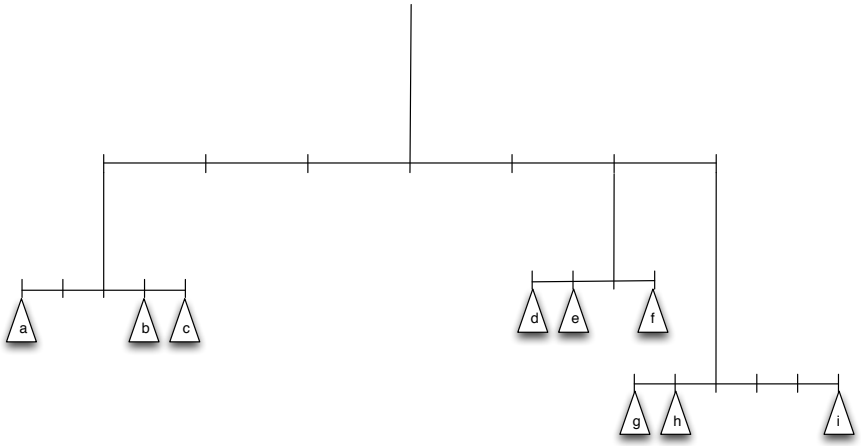

Figure 3.1: Balancing act: Where should the weights be placed?

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 solve 3 assignments:

1. select assignment 4.1 or 4.2– **deadline week 2**
2. select assignment 4.3 or 4.4– **deadline week 3**
3. select assignment 4.5 or 4.6– **deadline week 4**

You have to use JaCoP library together with Java or minizinc. At least one assignment has to be solved with Java or minizinc.

4.1 Lab assignment – Popsicle Stand[1]

Workforce management is central to efficient operations and good customer service. Proper scheduling of employees can mean the difference between profitability and business failure.

As the manager of a Popsicle stand, you are required to hire and set the weekly work schedule for your employees. The required levels for the week are as follows: Total employees required:

```
Monday = 5,
Tuesday = 7,
Wednesday = 7,
Thursday = 10,
Friday = 16,
Saturday = 18;
Sunday = 12.
```

Assume the same staffing requirements continue week after week.

Full-time employees work five consecutive days and earn $100 per day. Part-time employees work two consecutive days and earn $150 per day.

Question: What is the minimal weekly staffing cost you can achieve while meeting the required staffing levels?

[1] John Toczek http://www.informs.org/ORMS-Today/Private-Articles/February-Volume-39-Number-1/THE-PUZZLOR

4.2 Lab assignment – Urban Planning[2]

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 4.1: An example of residential area with commercial lots.

The 5x5 grid in depicted in Figure 4.1 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

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

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

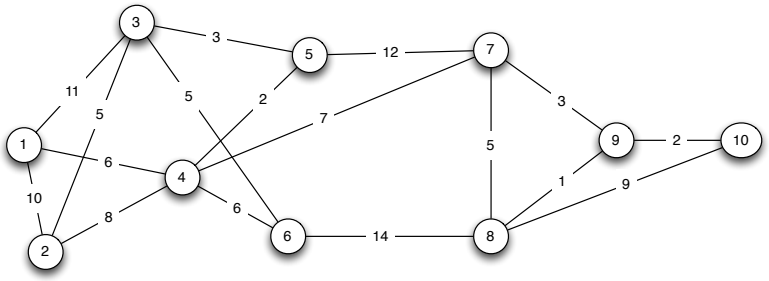

Figure 4.2: Map of the road between cities and their distance.

	City									
Fuel price	10	10	8	12	13	9	10	11	12	8

Table 4.1: Fuel prices.

4.3 Lab assignment – Full Tank

After going through the receipts from your car trip through Europe this summer, you realised that the gas prices varied between the cities you visited. Maybe you could have saved some money if you were a bit more clever about where you filled your fuel?

To help other tourists (and save money yourself next time), you want to write a program for finding the cheapest way to travel between cities, filling your tank on the way. We assume that all cars use one unit of fuel per unit of distance, and start with an empty gas tank.

Figure 4.2 shows the cities number from 1 to 10 and distances between them. Price of the fuel in each city is as presented in table 4.1. Model this problem and find the price of the cheapest trip from city 1 to city 10 using a car with tank capacity 10. What will be the price for this trip if the car has tank capacity 15?

Hint: You may consider using `Subcircuit` constraint for this assignment.

4.4 Lab assignment – Jobshop scheduling

Problem:

The jobshop scheduling problem requires to schedule number of jobs on available machines. Each job is a sequence of tasks which need to be executed on different machines (M). The duration of each task is D. The table below specifies a simple job shop scheduling problem, which has 6 jobs (each job containing the sequence of 6 tasks) and 6 machines (0, 1, 2, ..., 5). Formulate the problem using JaCoP solver and schedule all tasks while minimizing the length of the schedule.

Job	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6	
	M	D	M	D	M	D	M	D	M	D	M	D
1	2	1	0	3	1	6	3	7	5	3	4	6
2	1	8	2	5	4	10	5	10	0	10	3	4
3	2	5	3	4	5	8	0	9	1	1	4	7
4	1	5	0	5	2	5	3	3	4	8	5	9
5	2	9	1	3	4	5	5	4	0	3	3	1
6	1	3	3	3	5	9	0	10	4	4	2	1

Hint: This problem can be solved using, among other constraints, the `Cumulative` constraint.

4.5 Lab assignment – Auto Regression Filter

Problem:

The data-flow graph for the auto regression filter is depicted on Figure 4.3. It consists of 16 multiplications and 12 additions. These operations need to be scheduled on multipliers and adders. Write a program which will

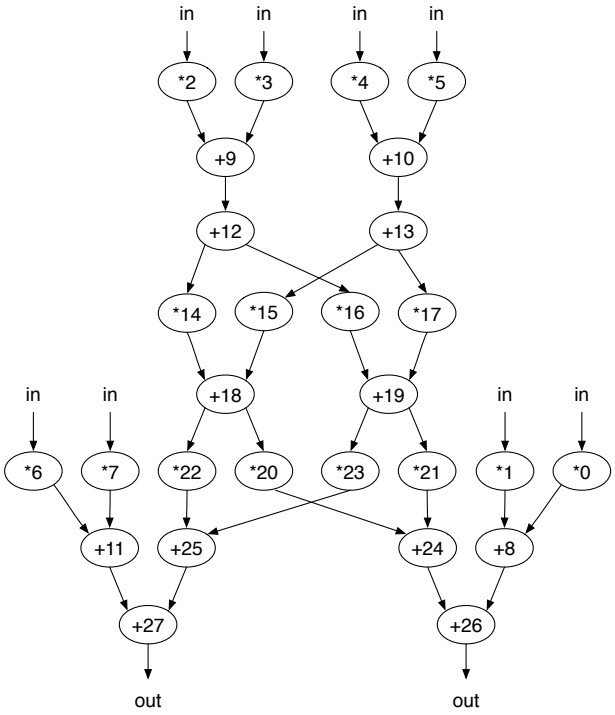

Figure 4.3: The operation graph representation of the auto regression filter.

optimize the schedule length for different amount of resources as specified in table 4.2. 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.

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

Table 4.2: Experimental results

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

4.6 Lab assignment – Task Scheduling

A task graph of nine tasks is depicted in the Figure 4.4. Each circle represents a task and an arc data transfer between tasks, i.e., a task can be executed when all its predecessors have been executed and transferred their data. These tasks can be scheduled on three different processors. Assume that you have available two processors from each group for task execution. The cost of processors as well as the execution time of each task on different processors are different and given in the table. The cost of a processor is counted once regardless of how many tasks are assigned to a processor, for example, if we use one processor to execute all tasks the cost for this solution will be equal the cost of the selected processor.

Assume also a simple scenario for data transfers. All data transfers between processors are done on point-to-point links. Data transfers between tasks assigned to the same processor are done be coping data in memory and they does not use point-to-point links. The point-to-point link has transmission time of one time unit and it contributes to the system cost with one “cost” unit. Note that transmission links can be reused, i.e., if there is a link between two processors it can be used for several transmissions but its contribution to the cost will be still one. Data transfers between tasks allocated to the same processor take zero time. Model this example for both scenarios using JaCoP and formulate an execution time minimization method as well as processor cost minimization.

Give the results in table 4.4.

Table 4.3: Execution times and costs for example 7.

		Execution time of tasks								
Processor	Cost	1	2	3	4	5	6	7	8	9
p1	4	2	2	1	1	1	1	3	-	1
p2	5	3	1	1	3	1	2	1	2	1
p3	2	1	1	2	-	3	1	4	1	3

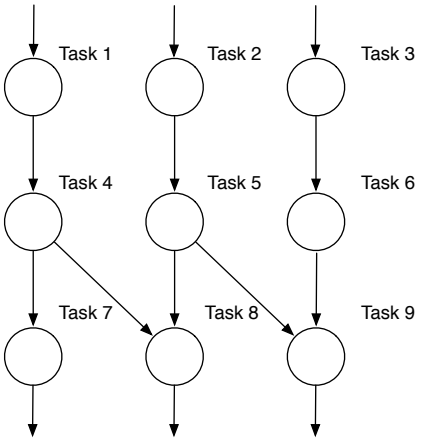

Figure 4.4: The task graph for exercise 7.

Resource-constrained optimization		
Cost limit	Time	Runtime [s]
15		
12		
8		
7		
5		

Time-constrained optimization		
Time limit	Cost	Runtime [s]
5		
6		
7		
8		
15		

Table 4.4: Experimental results.

5

Depth First Search example- deadline week 6

5.1 Lab assignment

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 described, for example in `http://mathworld.wolfram.com/GolombRuler.html`.

In this assignment you have to change the behavior of this search that it can select variables with smallest domains first and assigns maximum value to each variable first. Test your solution on the included example.