# AI 1: Lab session 3

## Constraint Satisfaction Problems

**Instructions**

- Answer all questions and ***formulate your answers concisely and clearly***.
- Read the book and use the information from the lectures. Read the questions carefully and feel free to ask for clarifications if something is not clear.
- You can use the helpdesk email `airug1819@gmail.com` to ask questions. Make sure you read the instructions to contact the helpdesk that are provided on Nestor. Only emails sent according to the instructions will be replied to.
- For the programming assignments, ***it is not allowed to use code supplied by others***. If we suspect plagiarism or collusion, the exam committee will be notified.

**Rules for submission**

- Submit, electronically via Nestor (Submissions section), a .zip file containing:
  1. ***All relevant source codes***, so that we can test them.
  2. A digital version of your team's report. ***Reports need to be written in LaTeX***. A template is available on Nestor.
- Comply with the deadline provided on Nestor. ***Deadlines are strict*** (penalties are applied for late submissions, see below).
- Supply the names and student numbers of all members of your team on the report. Also clearly ***write the learning community*** of each member in the group.

**Grading** Every assignment is graded by the teaching assistants based on grading schemes agreed with the lecturer. The grade of this lab assignment will count for ***10% of your final grade***. We ***subtract*** $2^{n-1}$ grade points for a submission that is between $n-1$ and $n$ days late ($n \geq 1$).

# Introduction

All the problems of this lab session must be solved using a CSP solver that was made by Jelle Bakker for his Bachelor Project in 2014/15. The solver solves generic CSPs using the standard backtracking algorithm augmented with the heuristics MRV (Minimum Remaining Values) and DEG (Degree heuristic). Moreover, the FC (Forward Checking) and ARC (Arc Consistency) propagation algorithms have been implemented. The 64-bit Linux executable of this solver can be found on Nestor.

The solver consists actually of two programs. The first is called `csp` and accepts on its command line a couple of flags and a CSP specification file. Use `./csp -help` to see the supported flags. The program converts a CSP specification written in a small specification language which allows the usage of arrays, alldiff() constraints, for all constructs etc. into an equivalent specification which does not support this syntactic sugar (the standard form as used in the lectures). The converted CSP in normal form is passed to the second program, which is the actual solver. In normal circumstances, the user does not notice this two-step process, since the program `csp` automatically starts the back end program. This two stage pipeline is depicted in the following figure.


It is important to realise that *the specification language is not a programming language*: there are no while/if/for-constructs. The language contains a `forall` construct, but it is used for specifying constraints with indexes (and not for iteration in a specific order). It is important to specify what you want to compute, not how you want to compute it. That remains the task of the solver. However, it definitely makes a difference how you specify a problem. One specification may yield solutions faster than an equivalent other specification.

A CSP specification file consists of 4 required sections (in this order):

- **variables section**: declaration of the variables of the CSP. For example
  `variables: a, b, c, d[3]: integer;`
  declares four variables: three scalars `a`, `b` and `c`, and a small vector `d` consisting of the elements `d[0], d[1], d[3]`.

- **domains section**: declaration of the domains. For each variable a finite domain must be given. A domain may be given as an explicit list, but also as ranges. For example
  `domains: a, b, c <- [0..9]; d <- [2, 3, 5];`
  specifies that `a`, `b` and `c` are digits, while each element of `d` can be either 2, 3, or 5.

- **constraints section**: : This section describes the constraints of the problem. These constraints must be written as arithmetical expressions/equations that evaluate to either true or false, e.g. `a + b = 5` evaluates to true if `a=2, b=3`. The supported operators in arithmetical expressions are the integer operators `+, -, *, ^, div, mod`, where `^` denotes exponentiation, `div` integer division, and `mod` the remainder by division. Moreover, the functions `max(a,b)`, `min(a,b)`, and `abs(a)` are available to compute respectively the maximum, minimum of `a` and `b`, and the absolute value of `a`. Note that the operators `max` and `min` can also be used to simulate the boolean operators $\vee$ and $\wedge$. The available

relational operators are `<`, `<=`, `=`, `>=`, `>`, `<>`, where `<>` denotes inequality.

The special constraint `alldiff(d)` is available to specify that all elements of the array `d` have different values. It can also be used to specify that a list of scalars differ, i.e. `alldiff(a,b,c)`. There is a `forall` construct available (which can be nested), that can be used for indexing. For example, the construct `forall (i in [0..2]) d[i] = i+1; end` specifies that `d[0]=1, d[1]=2, d[2]=3`. Also available are the boolean multi-argument functions `all` and `any`, that implement the conjunction and disjunction of their arguments (i.e., the implementation of $\forall$ and $\exists$).

- **solutions section**: This is a small section. It is used to specify the number of solutions that we are interested in (possibly all). For example
  `solutions: all`
  specifies that we want all solutions of the problem. The keyword `all` can be replaced by a positive integer (e.g. 1 means that the solver stops immediately after it found a solution).

Currently, the only type supported by the solver is `integer`. Booleans can be simulated using the convention used, e.g., in the programming language C: 0 represents `false`, while non-zero represents `true`. The program supports scalar variables and (multi-dimensional) arrays. Comments are supported, and start with the symbol # and run until the end of line (like `//` style comments in Java or C++).

**Note** The specification language used for the solver is quite expressive, and has been tested extensively. Nevertheless, bugs may be present. If you find a bug, please notify your TA and send a bug report to the helpdesk.

## Heuristic and propagation technique analysis

For each of the following problems, you are required to hand in an analysis why (a combination of) heuristics and propagation techniques (invoked by using flags) is beneficial (or not). The analysis should mention the optimal combination of these heuristics and propagation techniques, and explain why, considering the theory behind them.

Moreover, you are requested to answer any additional questions posed in the exercise. The problem description files (with extension .csp) for these problems can be downloaded from Nestor. You only have to run them (instructions on how to run these files are above).

### 1. Solving a small set of equations

Consider the following equations:

$$\begin{aligned} A + B &= 5 \cdot C \\ B + C &= 3 \cdot A \\ A \cdot B \cdot C &= 42 \end{aligned}$$

The content of the CSP description for this exercise is:

```
#The set of variables of the CSP
variables:
    A,B,C : integer;
#Here the domains are defined
domains:
    A,B,C <- [0..250];
#Here are the constraints:
constraints:
    A + B = 5*C;
    B + C = 3*A;
    A * B * C = 42;
# Here you can specify in how many solutions
# you are interested (all, 1, 2, 3, ...)
solutions: all
```

**Questions:**

**a)** Give the total number of solutions found by the CSP solver.

**b)** Analyze the heuristics and propagation techniques most effective for this problem.

## 2. Market

You probably remember the following exercise from the course *Imperative Programming*:

*You go to the market to buy oranges, grapefruits and melons. You are given the task to spend exactly 100 euros on the purchase of exactly 100 fruits. Given the prices of the three types of fruits (in cents), print all possible combinations that you can buy. The prices are non-negative integer numbers.*

This problem is modelled for you in `market.csp`

**Questions:**

**a)** Give the total number of solutions found by the CSP solver.

**b)** Analyze the heuristics and propagation techniques most effective for this problem.

## 3. Chain of trivial equations

Consider the problem described in `chain.csp`. This problem consists of 26 variables, where each variable has the domain `[0..99]`. The equations `A=B; B=C; C=D; .......; Y=Z;` model that all variables must have the same value.

**Questions:**

**a)** How many solutions exist?

**b)** Now consider the problems described in `chainA.csp` and `chainZ.csp`. Does the choice between the added constraints make a difference? Explain your answer.

**c)** Analyze the heuristics and propagation techniques most effective for this problem.

## 4. Cryptarithmetic puzzles

In *cryptarithmetic puzzles* some words (consisting of at most 10 letters in total) are used in an arithmetic expression. Each letter stands for a unique digit. The same letter stands for the same digit throughout the puzzle. Different digits are different letters. There are no leading zeros.

Consider the famous *cryptarithmetic puzzle* `SEND+MORE=MONEY`, which has the unique solution `D=7, E=5, M=1, N=6, O=0, R=8, S=9, Y=2`. Run the given problem description file for the CSP solver to verify this answer. Also run the solver for the following puzzles.

```
UN + UN + NEUF = ONZE
ONE + NINE + TWENTY + FIFTY = EIGHTY
I + GUESS + THE + TRUTH = HURTS
```

**Questions:**

Which combination of heuristics and propagation techniques are effective for each of the puzzles? Explain your results.

## 5. Finding the first 20 primes

Given is a CSP specification file that computes the first 20 prime numbers.

**Questions:**

Which flags need to be supplied to the solver to get an answer (in reasonable time). Explain your answer.

## 6. Solving Sudokus

In case you do not know the rules for sudokus, visit `https://en.wikipedia.org/wiki/Sudoku`.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 8 | 6 | 3 | 2 | 4 | | |
| | 4 | | | | | | 1 | |
| 5 | | | 9 | | 4 | | | 6 |
| 8 | | | | | | | | 5 |
| 6 | | | | | | | | 4 |
| 1 | | 7 | | | | 9 | | 2 |
| 4 | | | 7 | 5 | 1 | | | 3 |
| | 6 | | | | | | 2 | |
| | | 5 | 8 | 2 | 6 | 7 | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | 1 | 2 |
| | | | | | | | | 3 |
| | | 2 | 3 | | | 4 | | |
| | | 1 | 8 | | | | | 5 |
| | 6 | | | 7 | | 8 | | |
| | | | | | 9 | | | |
| | | 8 | 5 | | | | | |
| 9 | | | | 4 | | 5 | | |
| 4 | 7 | | | | 6 | | | |

**Questions:**

**a)** Solve the two sudokus using the CSP solver.

**b)** Compare the performance of the solver for the two puzzles.

**c)** Analyze the heuristics and propagation techniques most effective for the two sudokus.

## 7. $n$-queens problem (again)

Test CSP specification files for the $n$-queens problem for $n \in \{4, 5, 6, 7, 8, 9, 10\}$.

**Questions:**

**a)** For each of these problems, how many solutions exist? You may consider rotation, mirroring and flipping of solutions as unique solutions.

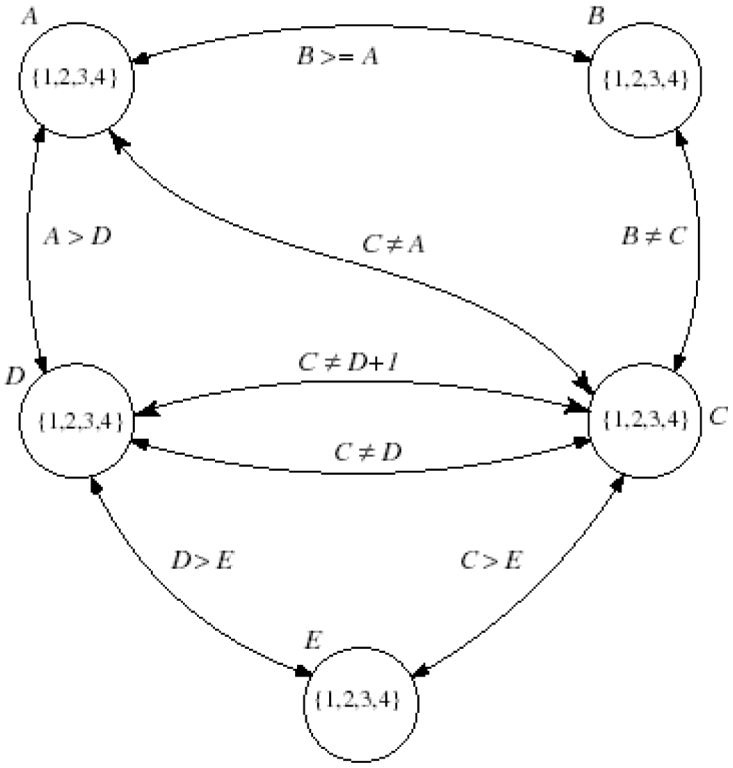
**b)** Provide the analysis of heuristics and propagation techniques for all problems.

## Writing CSP descriptions

For the following three exercises you are required to write your own CSP descriptions. Use the previous examples and the explanation given earlier to make them. Answer all the questions in the exercises as well as give an analysis of the flags used.

### 1. Constraint graph

Consider the following constraint graph:

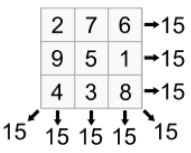



**Questions:**

Determine all solutions.

### 2. Magic squares

A *magic square* of size $n$ is an arrangement of the distinct integers $1, 2, \ldots, n^2$ in a square grid, where the numbers in each row, and in each column, and the numbers in the main and secondary diagonals, all add up to the same number. A magic square has the same number of rows as it has columns. An example of a magic square with size $n = 3$ is given in the following figure (source Wikipedia):



**Questions:**

How many magical squares exist for $n = 4$?

## 3. Boolean satisfiability (SAT)

Find values for the boolean variables $x_1, \ldots, x_5$ such that the following boolean expression evaluates to *true*:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_5) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_3 \vee x_5) \wedge (x_1 \vee \neg x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_5) \wedge (\neg x_3 \vee x_4 \vee \neg x_5)$$

**Questions:**

How many solutions exist?