APLAI Assignment 2016-2017

February 27, 2017

1 Introduction

- The APLAI assignment consists of two tasks. You work in groups of two: please put your group on the wiki of APLAI (see the Course Documents on Toledo).
- Due date: **Wednesday 07/06/2017, 13h00** . You email your report (a pdf file), programs and README file to gerda.janssens@cs.kuleuven.be.
- During the exam period, there is an oral discussion for each group on one of the following dates: 21/06/2016, 26/06/2016, 27/06/2017 and 28/06/2016. Further arrangements via Toledo.
- The grading is based on the report (content, clarity, conciseness), the quality of the code (style, readability and documentation) and the oral discussion.
- As the assignment is part of the evaluation of this course, the Examination Rules of the KULeuven are applicable. You should follow the rules of academic integrity. Write your own solutions, programs and text. Run your own experiments. When receiving assistance, make sure it consists of general advice that does not cross the boundary into having someone else write some of the actual code. It is fine to discuss ideas and strategies, but be careful to write your programs on your own. Do not give your code to any other student who asks for it and do not ask anyone for a copy of their code. Similarly, do not discuss algorithmic strategies to such an extent that you end up turning in exactly the same code. Always mention your sources.

  The previous holds for contacts in person but also for example for online forums.
- The APLAI WIKI page contains information about the installation and use of the ECLiPSe, CHR and Jess systems. THE APLAI WIKI page also contains additional ECLiPSe and CHR exercises.
- Guidelines for the report:
  - A typical report (for Task 1 and 2) consists of 20 (maximum 25) pages using a font and layout similar to this text.

- It has an introduction and conclusion. In the conclusion you give a critical reflection on your work. What are the strong points? and the weak points? What are the lessons learned?
- You can include some code fragments in your report to explain your approach. If you do, make a good selection. Do not include the complete code in the report as the code is in the program files.
- When you run your experiments, do not just give the time and/or search results, but try to interpret the results. Include the benchmark problems given on Toledo in your experiments.
- Add an appendix that reports on the workload of the project and on how you divided and allocated the tasks in this project.

2 Task 1: Sudoku

2.1 Task 1.A Viewpoints and Programs

The classical viewpoint for Sudoku states that all numbers in a row must be different, that all numbers in a column must be different, and that all numbers in a block must be different.

- Give a **different** viewpoint.
- How do you deal with the block constraint in the alternative viewpoint?
- What are the criteria you can use to judge whether a viewpoint is a good one or not?
- Discuss the possibilty of using **channeling** constraints between your alternative and the classical viewpoint? [1]
- For 2 out of the 3 systems (ECLiPSe, CHR and Jess): program the 2 viewpoints and if possible give and explain the code for the channeling.

2.2 Task 1.B Experiments

On Toledo you find a set of Sudoko puzzles (see `sudex_toledo.pl`). You should include them in your experiments, using the same names to refer to the puzzles, but you can also include your own favorite Sudoku puzzles.

For the experiments, you should run your programs for the different viewpoints and if possible a version with channeling constraints. Collect information and report about the run-times and the search behaviour (number of backtracks).

- For ECLiPSe you should experiment with the following settings: input_order and first_fail as variable heuristics, the default indomain as value heuristic and alldifferent/1 from the libraries ic and ic_global.
  Discuss the impact of the variable heuristics and of the alldifferent versions.

[1] Channeling as defined in the APLAI course and also in section 1.9 of `http://www-module.cs.york.ac.uk/copr/handbook-modelling.pdf`.

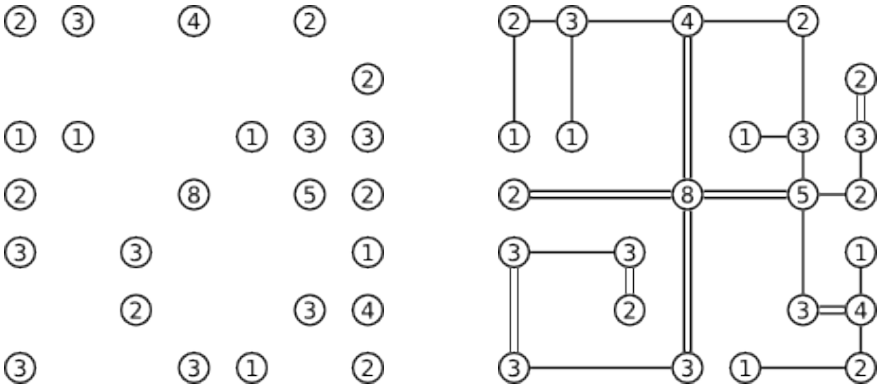

Figure 1: A Bridges puzzle and its solution.

- Select two Sudoko puzzles for which you observe substantial differences using different settings in the previous experiments. Try to explain them.
- For CHR, compare the two viewpoints and discuss the impact of channeling (if possible). Which heuristics are implemented in your programs?

3 Task 2: Hashiwokakero

See also `https://en.wikipedia.org/wiki/Hashiwokakero`.

3.1 Problem definition

Hashiwokakero, also called *Bridges* is a logic puzzle in which different islands have to be connected by bridges. A bridges puzzle consists of a square grid in which some numbers are placed. Squares on which a number is placed are referred to as *islands*. The goal of the puzzle is to draw bridges between islands subject to the following restrictions.

- Bridges can only run horizontally or vertically.
- Bridges run in one straight line.
- Bridges cannot cross other bridges or islands.
- At most two bridges connect a pair of islands.
- The number of bridges connected to each island must match the number on that island.
- The bridges must connect the islands into a single connected group.

Figure 1 contains an example of such a puzzle, and its solution.

The task is to write two programs, the first using ECLiPSe and the other using CHR or Jess, that solve instances of these puzzles.

3.2 Examples

On Toledo you will find a set of problems (see `puzzles.pl`).

Each `puzzle(Id,S,Islands)` fact defines the input of one problem: its identifier `Id`, the size `S` (this is the width and height), and the list of islands `Islands`. Each island takes the form `(X, Y, N)` where `X` is the row number, `Y` is the column number and `N` the number of bridges that should arrive in this island. The origin of the grid starts at the top left corner, the indices start from (1,1).

3.3 Tasks

3.3.1 Task 2.A: Implementation in ECLiPSe

1. Implement in ECLiPSe a **basic solver** that finds a solution for the Bridges problem as defined above. You can (but are not obliged to) start from the partial solution of jschimpf on `http://stackoverflow.com/questions/20337029/hashi-puzzle-representation-to-solve-all-solutions-with-prolog-restrictions`

   Address the following questions in your report:

   - Explain which constraints you use and how they are expressed in your program. Are the constraints active or passive ones?
   - Discuss the impact of the different search strategies.
   - For your experiments, you can use the instances in `puzzles.pl`.

2. Propose at least three additional improvements (e.g. redundant constraints). Inspiration can be found on `http://www.conceptispuzzles.com/index.aspx?uri=puzzle/hashi/techniques`. Include these improvements in your solver and discuss their impact.

   Discuss how you modelled the connectivity constraint in your solution. Do the additional improvements have an impact on the connectivity constraint?

3.3.2 Task 2.B: Implementation in CHR/Jess

In order to program this problem in CHR or Jess, you will have to encode more things yourself such as constraint propagation and search. Implement at least two different versions of propagators for the connectedness constraint.

Your report should at least describe the following steps:

1. Choose and explain a suitable representation of the data.
2. Choose and explain a suitable representation of the constraints.
3. Describe how you deal with constraint propagation, and what kind(s) of propagation you support. Are the constraints passive or active? Try to have active constraints if possible. In particular, discuss how you handle the constraint that all islands need to be connected.
4. Implement a basic solver that finds a (first) solution. What kind of search is it using?

5. Propose some additional constraints that may speed up solving. Implement at least 2 of them and give example puzzles for which you have/expect a speed up.

6. Describe the effect of these additional constraints on the performance of your implementation. In particular, do they have an impact on the connectivity constraint?

If you use CHR for this part, you can have a look at the CHR program for the N-queens problem (given on Toledo). It uses a finite domain solver.

General tips:

- CHR and ECLiPSe are both based on Prolog. You may be able to reuse some code between both tasks.
- For CHR, use the SWI-Prolog version and not the version in ECLiPSe.
- If your CHR program makes SWI-Prolog run out of memory (e.g. Global Stack), restart the SWI system. Also after reloading your CHR program file a number of times, it is a good idea to restart SWI anyway.
- The stackoverflow solution referred to above contains a predicate that can be used to visualize the puzzle and its solution for Eclipse and CHR.