APLAI Assignment 2017-2018

February 26, 2018

1 Introduction

The APLAI assignment consists of two tasks.

1.1 Practicalities

- You work in groups of two: please put your group on the wiki of APLAI (see the Course Documents on Toledo).
- Make sure you divide the work such that each of you is involved in both tasks and each of you works with more than one programming tool.
- Due date: **wednesday 6/6/2018, 12h00** .
  You submit your report (a pdf file), programs and a README file on Toledo (details follow later).
- During the exam period, there is an oral discussion for each group on one of the following dates: **20/6, 25/6, 26/6 and 27/6**. 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 this assignment is an important part of the evaluation of this course, the Examination Rules of the KU Leuven 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 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.
- The previous holds for contacts in person but also for example for online forums. This is not an open source project, i.e., you are also not allowed to put your code openly available on the web. If you want to use git, do not use public GitHub repositories.
- Use a scientific approach and mention your sources.

- 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:
  - The report consists of **maximum** 10 pages in total (using a font and layout similar to this text). This limit is **strict**: if your report is too long you can not succeed for this course.
  - In the conclusion of the report you give a critical reflection on your work. What are the strong points? and the weak points? What are the lessons learned?
  - Make a good selection of the code fragments you put in your report. Use them to explain your approach. 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. This appendix could be on page 11.

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**. Make sure you can define **channeling** constraints between your alternative and the classical viewpoint [1].
- Program the two viewpoints and the channeling between them for ECLiPSe and for either CHR or Jess.
- Using a different viewpoint has an impact on the modelling of the Sudoku constraints. Which of the constraints became the most challenging one in your alternative viewpoint? Explain your solution for it and give the relevant code snippets of both implementations.
- Explain your approach for channeling and illustrate it by giving the relevant code snippets of both implementations.

[1] Viewpoint and channeling as defined in the APLAI course and also in sections 1.8 and 1.9 of `http://www.dcs.gla.ac.uk/~pat/cpM/contrib/bms/Smith.pdf`

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!

1. ECLiPSe
   (a) Start with running your experiments for the original viewpoint with the default indomain as value heuristic and discuss the impact of the following settings: input_order versus first_fail as variable heuristics and alldifferent/1 from the libraries ic and ic_global.
   (b) Report on the run-times and the search behaviour (number of backtracks) for the given puzzles.
   (c) Explain the impact of input_order versus first_fail using ic on puzzle *extra2*.
   (d) Explain the effect of ic_global on the run-times and the number of backtracks for sudowiki_nb49.
   (e) Repeat the same experiments as in (a) with your alternative viewpoint in ECLiPSe. What are the changes you observe? Try to explain them.
   (f) What is the effect of adding the channeling constraints? Try to explain it.
2. CHR
   - Describe the variable and value heuristics implemented in your programs.
   - Compare the two viewpoints for the given Sudoku puzzels and discuss the impact of channeling.

3 Task 2: Hitori

See also `https://en.wikipedia.org/wiki/Hitori` and `http://www.menneske.no/hitori/eng/`.

3.1 Problem definition

Hitori, is a logic puzzle in which numbers in a grid have to be blacked out.

A Hitori puzzle consists of a square grid of size S with at each position a number between 1 and S.

The goal is to black out a number of cells such that:

- no row or column has more than one occurrence of a given number,
- no two black cells are (horizontally or vertically) adjacent,
- all white cells (those that are not blacked out) are connected; that is: for every two white cells c_1 and c_2, there exists a path from c_1 to c_2 consisting of only white cells where every two subsequent cells in the path are (horizontally or vertically) adjacent.

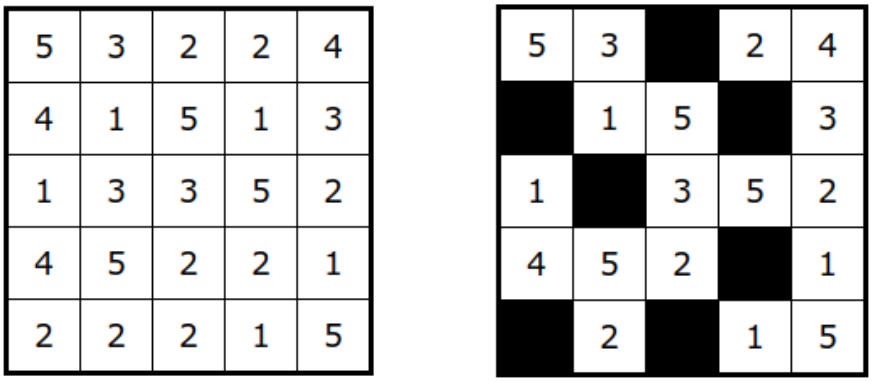

Figure 1: A Hitori puzzle and its solution.

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`). All of these puzzles come from the site `http://www.menneske.no/hitori/eng/`; they vary in size and difficulty. The file `puzzles.pl` also contains links to the solutions of the various puzzles, so that you can verify whether your program finds the right answer.

Each `puzzle(Id,S,P)` fact defines the input of one problem: its identifier `Id`, the size `S` (this is the width and height), and the input grid `P`. The input grid is a matrix of width and height `S`, i.e., a list of `S` lists of size `S`. At each point in the matrix is a single number between `1` and `S`. For instance, the puzzle with identifier `1` is the puzzle depicted in Figure 1 above.

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 Hitori problem as defined above.

   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 how you modeled the connectivity constraint in your solution.
   - Discuss the impact of different search strategies.
   - For your experiments, you should include the instances in `puzzles.pl`.

2. Propose at least four additional improvements (e.g. redundant constraints)[2]. Inspiration can be found on `http://www.menneske.no/hitori/methods/`

[2]Redundant or implied constraint as defined in the APLAI course and also in section 1.7 of `http://www.dcs.gla.ac.uk/~pat/cpM/contrib/bms/Smith.pdf`.

`eng/`. At least two of these improvements should have something to do with the connectivity constraint (i.e., locally ensure connectivity).

Include these improvements in your solver and discuss their impact.

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 white cells need to be connected. **Bonus:** you will get bonus points if you implemented the connectivity constraint as an active constraint.
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, and implement a at least 2 of them. At least one of them should have something to do with the connectivity constraint (i.e., locally ensure connectivity).
6. Describe the effect of these additional constraints on the performance of your implementation. In particular: what did you expect and how do the different versions (of the connectivity constraint) you implemented perform?

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.