# APLAI Assignment 2018-2019

March 19, 2019

## Introduction

### 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 all the tasks and each of you works with more than one programming tool.
- Due date: **tuesday 3/6/2019, 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: **18/6. 19/6, 24/6, 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 12** 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.

# 1 Task 1: Sudoku

## 1.1 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 Handbook-Smith.pdf as available on Toledo or on `http://www.dcs.gla.ac.uk/~pat/cpM/contrib/bms/Smith.pdf`

### 1.2 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.

## 2 Task 2: Hashiwokakero

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

### 2.1 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.

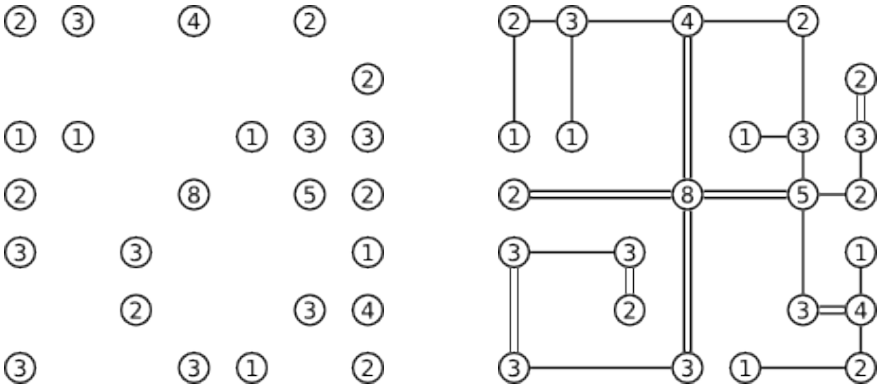

Figure 1: A Bridges puzzle and its solution.

- 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.

## 2.2 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).

## 2.3 Tasks

### 2.3.1 ECLiPSe

Start from the partial solution posted on Toledo. This solver covers all of the constraints mentioned above, but not the *connectedness constraint*, which states that in a correct solution, all the islands should form a single connected group (which means that any island can be reached from any other island).

1. Edit the partial solution to work with our input format.
2. Extend the partial solution with support for the connectedness constraint. You must implement this as a constraint solving problem, it is *not* sufficient to simply filter out unconnected solutions after the labeling step. See Section 2.4 for some hints on how to get started with this.
3. In your report, describe how your solution covers each of the constraints of the Hashiwokakero puzzle (including the ones that were already covered by

the partial solution, you should show that you understand why it works). Are the constraints active or passive?

4. Experiment with some different search strategies, and give one example of a strategy that works well, and one example of one that works very badly. Explain the cause of their performance characteristics.

5. For your experiments, you can use the instances in `puzzles.pl`.

#### 2.3.2 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.

Your report should at least describe the following issues:

1. Choose and explain a suitable representation of the data.

2. Implement a **basic solver** that finds a (first) solution. What are the main CHR constraints in this solver?
   You are allowed (but not obliged) to enforce connectedness by filtering out unconnected solutions after labeling.

3. Describe how this solver deals with the constraints of Hashiwokakero and their propagation. Are the constraints passive or active? Try to have active constraints for the Hashiwokakero constraints (at least for the first five of the six given in the Problem definition).

4. What kind of search is used in the basic solver?

5. Consider the hints for solving Hashiwokakero on
   `https://www.conceptispuzzles.com/index.aspx?uri=puzzle/hashi/techniques`.
   **Additional constraints** may speed up solving. Select two additional constraints and give example puzzles for which you expect a speed up.

6. Add the two additional constraints to your basic solver and discuss their impact on the performance. 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.

### 2.4 Connectedness constraint

We recommend an approach where you consider the islands and bridges as, respectively, the nodes and edges in a *flow network* (`https://en.wikipedia.org/wiki/Flow_network`). One node is arbitrarily chosen to be the sink node of the flow network. Every other node is a source that generates one unit of flow. The islands form a single connected group if and only if there is a way to assign flow to the edges in such a way that the total incoming flow at the sink node is equal to the number of islands minus one.

Flow can be represented similarly to the way the partial solution represents bridges: create an additional four constraint variables for each grid cell, one for each cardinal direction (for example `FlowNorth`, `FlowSouth`, `FlowEast`,

`FlowWest`). The value of each variable corresponds to the amount of flow flowing over that cell in the direction associated with the variable. Then we impose the following constraints:

1. You can't have any flow in a cell which does not contain a bridge or island.
2. A bridge cell does not generate net flow, so the net flow on every bridge cell should be zero (for example, if there is a flow of 2 going North on a bridge cell, then the same cell should have a flow of $-2$ going South).
3. If a bridge cell has a flow of $n$ going in a given direction, then the neighboring cell in that direction should have a flow of $-n$ going the opposite direction. For example, if a bridge cell has a flow of 2 going North, then the *cell to the North* should have a flow of $-2$ *going South*.
4. Every non-sink island generates 1 unit of flow, so the total amount of flow leaving any non-sink island should be one higher than the amount of flow arriving on the island.
5. The net flow arriving at the (unique and arbitrarily chosen) sink island should be equal to the total amount of flow generated, that is, the number of islands minus one.

### 2.5 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.

## 3 Task 3: Scheduling meetings

You are a student representative with as mission to convince the Minister of Education to change the exam regulations in codex of higher education. You will need the support and the help of several members of the universities and the ministry in order to succeed in your mission. Your strategy is to meet all influential people to get to know the Minister's background and habits and to ask these people to convince her to accept your proposal.

You must schedule each meeting with these people and the final (last) meeting with the minister. Each meeting will take a number of days, some people do not want to have meetings over the weekend and meetings can not be interrupted. In order to be able to meet some of the people, you need to have completed the meeting with some other person: these precedence rules have to be respected. All these people have a rank in the society.

The aim is to complete all the meetings as soon as possible, but also to try to avoid as much as possible to meet with people of higher rank before people of lower rank. Note that the most important thing is to minimize the end time of the meetings, and secondly to minimize the number of rank violations.

Consider the following example. You will be meeting with Dean Steyaert, Wouter Beke, Rector Sels, Rector Van Goethem and finally Minister Crevits. You need to have your meeting with Steyaert, before you can meet Sels. Your final meeting is with Crevits. No meeting will contain a Saturday or a Sunday.

## 3.1 Tasks

1. Write in ECLiPSe the following predicate meeting/9 to schedule your meetings.

```
test1(Start,End,Viol) :-
     NbofPersons = 5,
             % the persons are represented by a number
             %    1 for Steyaert, 2 for Wouter Beke, 3 for Sels,
             %    4 for Van Goethem and 5 for Crevits
             % the last meeting is always with person NbofPersons
    Durations = [](1,2,3,4,5),
                 % Durations[I] is the duration of the meeting with person i
     OnWeekend = [](0,0,0,0,0),
                 % can the meeting with this person be in a weekend (0:no,1:yes)
     Precs= []( [](1,3)),
                 % precedences between meetings (in this example only 1)
                 % not including the "final meeting" constraints
     StartingDay = 5,
                 % which day of the week is day 0 in the schedule
                 % 0 Monday, 1 Tuesday, ..., 6 Sunday
     Rank = [](1,2,3,4,5),  % rank of the person
     meeting(NbofPersons, Durations, OnWeekend, Rank, Precs,
                  StartingDay, Start, EndTime , Viol).

finds a solution with
Start = [](2, 9, 11, 3, 16)
EndTime = 21
Viol = 2
```

Note that in this example no meeting is going on on day 0 and day 1 of the schedule, because in this example no meeting can be put on a Saterday or a Sunday. The first meeting is with peron 1 (Steyaert) on day 2 (Monday), then with person 4 (Van Goethem) on day 3 (Tuesday till Friday). In the next week you meet person 2 (Beke) on day 9 (Monday till Tuesday) and person 3 (Sels) on day 11 (Wednesday till Friday). And finally, you meet person 5 (Crevits) on day 16 ( Monday till Friday). The end of the schedule is on day 21, with two rank violations: person 4 (rank 4) is visited before person 2 (rank 2) and person 3 (rank 3).

Some more examples:

```
test1b(Start,EndTime , Viol) :-
     NbofPersons = 5,
     Durations = [](7,2,3,4,5),
     OnWeekend = [](1,0,0,0,0),
     Precs= []( [](1,3)),
     StartingDay = 5,
     Rank = [](1,2,3,4,5),
     meeting(Mode,NbofPersons, Durations, OnWeekend, Rank, Precs,StartingDay,Start, EndTime , Viol).

finds a solution with
```

```
Start = [](0, 9, 11, 16, 23)
EndTime = 28
Viol = 0

test1c(Start,EndTime , Viol) :-
     NbofPersons = 5,
      Durations = [](7,2,3,4,5),
     OnWeekend = [](0,0,0,0,0),
     Precs= [](  [](1,3)),
     StartingDay = 5,
     Rank = [](1,2,3,4,5),
     meeting(Mode,NbofPersons, Durations, OnWeekend, Rank, Precs,StartingDay,Start, EndTime , Viol).

fails as no solution exists
```

2. What are your constraints for the weekend requirements?

3. What is the cost you minimize?

4. Your meetings can not overlap in time. Consider two versions of meeting/9: one in which you make all required non-overlap constraints of the meetings explicit in your program, and one in which you use a global constraint from the library `ic_edge_finder`. (in the same way as you can use one all_different/1 constraint for a list of variables or specify all individual \=/2 constraints between the variables in the list).
   Compare the performance of the two versions by running the given tests found in the Toledo file benchmarksMeeting.ecl.