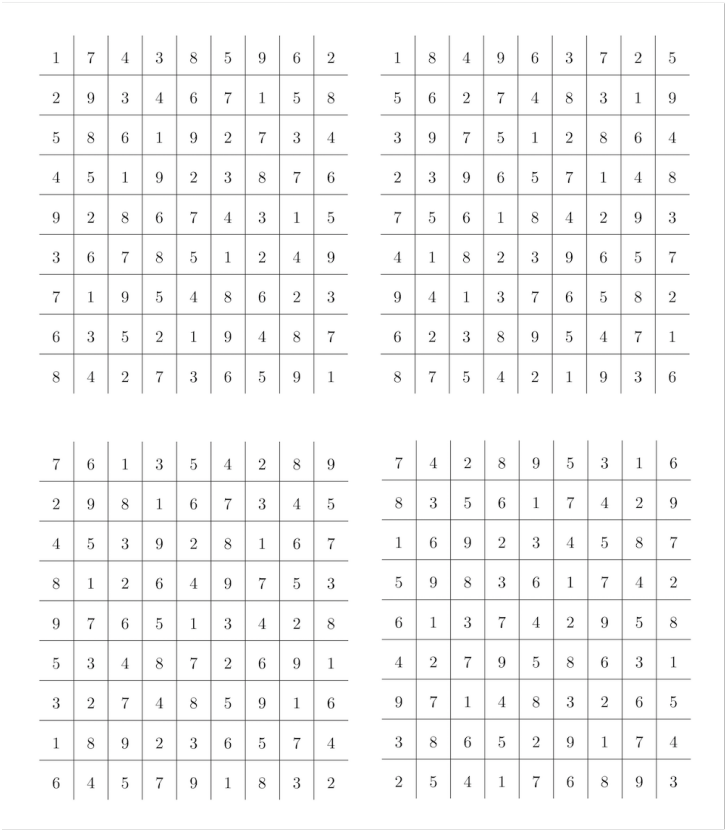
Advanced Programming Languages for A. I.

Michaël Dooreman & Bruno Vandekerkhove

Attention: Your report still contains 1 instructions or comments. Make sure to delete all of them for the final version. (Run `pdflatex` at least two times after removing them to make sure this warning disappears.)



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

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

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

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

Academic Year 2019

H02A8A: Advanced Programming Languages for A. I.

Contents

In the past decades much research has been done on the characterization and resolution of constraint satisfaction - and constraint optimization problems. This report discusses three challenges ; Sudoku puzzles, Hashiwokakero and a scheduling problem.

1 Sudoku

Sudoku[1] is a well-known puzzle game which needs no introduction. It is typically modelled as a constraint satisfaction problem through the use of `all_different` constraints on rows, columns and blocks. Such global inequalities tend to improve upon the use of binary inequalities. The constraint generating code[1] is fairly trivial and needn't be detailed here.

There are several other ways one could model Sudoku. The widely cited study by Helmut Simonis[REF] and subsequent studies provide some ideas. Four *'dual'* models, two approaches based on a boolean characterisation, a combination of models provided by Laburthe[REF], a model enforcing the singular occurrence of every value in every row, column and block, as well as a model with nothing but channeling constraints were considered[2]. Tests were run on the provided puzzles[3] as well as some minimum puzzles provided by Gordon Royle. These are puzzles with a minimal amount of pre-filled cells (17 to be precise[McGuire, 2014]), which does not mean that they are harder to solve.

The dual models hold a $N \times N$ array with all the decision variables. Whereas in the classic viewpoint the rows, columns and values of this array correspond to those of the input puzzle, every one of the four dual models changes their roles. The first two switch the role of rows or columns with those of values. In the third model every row and column of the array corresponds to a block and a position. In the fourth dual model every row represents a block, every column a value and every value a position within a block. For each of them it was harder to implement the necessary constraints, usually necessitating the use of auxiliary variables together with appropriate channeling constraints.

In one of his works Laburthe discusses various rules that can be used to resolve Sudoku puzzles, after which he details three models that he associates with the rules. He ends up proposing a model for every level of difficulty of the input puzzle. An attempt was made at implementing his recommendation for 'difficult' puzzles. It decreased the average number of backtracks but increased the runtime.

[1] `ECLiPSe` and `CHR` implementations are available in `/sudoku/model/classic.pl` and in `/sudoku/chr/model/classic.pl`.
[2] These are implemented with ECLiPSe in the `/sudoku/eclipse/model/` directory, and some CHR versions are in `/sudoku/chr/model/`.
[3] `/sudoku/benchmarks/benchmarks.pl` provides automatic benchmarking code.

The boolean models include the natural combined model [REF] and a more intuitive characterisation resembling an integer programming - or a SAT model [REF] (using `ic_global:occurrences/3` instead of sums, disjunctions and conjunctions). Both of them have $N \times N \times N$ boolean variables b_{rcv} which are true if the cell at row r and column c holds the value v. The natural combined model was cumbersome to implement and performed badly. It was introduced together with an algorithm after which it was tailored, and a constraint for unequality of lists isn't really supported by ECLiPSe[4].

As pointed out by Rossi it is usually not recommended to use a boolean model when integers can be used instead [REF ROSSI].

The last two models were found to be the most performant. The first makes use of the previously mentioned `occurrences/3` constraint to make every value occur just once in every row, column and block.

The second one generates nothing but channeling constraints. It has been demonstrated that this can provide good results despite such constraints being less 'tight' than `all_different` constraints[REF ROSSI]. When Dotu discussed it he was considering QuasiGroups[REF]. This was extended[5] to Sudoku puzzles by making use of three instead of two dual models (since blocks need to be considered as well). The variant in which channeling constraints between all models (one primal, three dual) are generated performed better than the one in which only channeling constraints between the primal and every dual model are applied. These variants are analogous to what Dotu referred to as *'trichanneling'* and *'bichanneling'*.

1.1 Experiments

Number of backtracks and running time for most of the models are displayed in table 1. Removal of *'big'* (`all_different`) constraints in the classic model[6] led to an increase in runtime as predicted by Demoen[REF].

One interesting observation is that the two most performant models also have the same number of backtracks. One of them is the model with nothing but channeling constraints, the other uses the `occurrences/3` constraint. This constraint enforces arc consistency.

» Maybe do some redundant constraints (those two from Simonis) with the note that the same/2 constraint does exist (http://eclipseclp.org/doc/libman/libman022.html) but now the focus lies on CHR and understanding the whole thing to explain results. Probably good enough to use dual3 for the experiments because it makes channeling easier.

» Two tables: one with all results with ic:alldifferent and just fixed heuristics (input_order). Then a table for ic_global + heuristics for the chosen 2 models (channeling and dual3 or dual4), this time each puzzle separately. Then a small discussion of the results. Should be 3 pages in total including TOC. Note that small discussion of CHR needs to be done too.

» Maybe for those puzzles that do better with input_order show that mirroring the puzzle leads to slower runtimes (you could do this for all puzzles, write procedure swap(Puzzle) for it). Note that value heuristics shouldn't matter much as the 'max' or 'min' isn't really meaningful in Sudoku (you can use '9,8,7, ...' instead of '1,2,3,...' for all we know

2 Hashiwokakero

TODO: Task 2

3 Scheduling Meetings

The last challenge is the scheduling of some meetings, taking into account the preferences of the various persons involved. A constraint optimization problem where the cost is a function of the end time of the last meeting and the number of '*violations*' (people of lower rank having their meeting after that of people of higher rank).

[4] A custom-made implementation as well as the `~=/2` constraint which checks if two terms can be unified were tried.

[5] The code lies in `sudoku/model/channeling.pl` in which a flag called `extended` can be used to opt for one of two variants.

[6] In his study Demoen gives several *Missing(6)* examples, models in which 6 of the `all_different` constraints are removed. *Missing(7)* models aren't Sudoku, and because of the stark rise in number of backtracks no further experimentation with the removal of '*small*' constraints was done. The `eliminate_redundancy` flag can be used to toggle redundancy elimination on and off.

This number of violations is of secondary importance.

Weekend constraints are generated first. If a person doesn't want to meet on weekends then his or her meeting is not allowed to overlap with the first weekend that follows :

$$((S + StartingDay) \texttt{ mod } 7) + D < 5$$

In the above constraint S and D represent the start and duration of the person's meeting. Making direct use of `mod/3` leads to an instantiation error, necessitating the use of an auxiliary variable representing the result of the modulo operation [CODE].

Precedence constraints and constraints assuring that no meetings overlap are generated last. The corresponding code is fairly trivial[7]. The fact that the meeting with the minister should come last is equivalent to adding $N - 1$ precedence constraints with N the total number of persons.

The cost function is defined as $(M \times E) + V$ where M is the maximum number of rank violations, E is the end time of the meeting with the minister and V is the actual number of rank violations for a given solution. This ensures that whenever two solutions have a different E, the solution with the smallest E will have the lowest cost (whatever the number of violations V). Yet if two solutions have the same end time E, then it's the number of violations V that will determine what solution is best. Code is displayed in [CODE].

Some implied constraints were added to increase performance. In case two persons have a different rank but the same meeting duration and weekend preferences, a corresponding order on their start times can safely be imposed. This mustn't override the precedence constraints. The respective implementation is displayed in [CODE].

Table ??? shows the runtime for each benchmark. Two versions are considered ; one ensures that no two meetings overlap by imposing a `(`$S_1{+}D_1 \leq S_2$ `or` $S_2{+}D_2 \leq S_1$`)` constraint for every such pair, the other version uses a global version of these same constraints provided by the `ic_edge_finder` library. It's clear that the global version outperforms the other one.

Instead of making use of implied constraints one can also tinker with the various heuristics provided by the `search/5` procedure. Some of those lend themselves to some benchmarks but not to others[8].

The `indomain_min` heuristic performed better than `indomain_max` as it is an optimisation problem after all, meaning that selecting the minimum starting time selects solutions with a smaller cost first. A solution with a lower cost will prune the search tree more.

[7] Note that all code for this third challenge can be found in `/src/scheduling/scheduling.pl`.

[8] The `occurence` variable heuristic speeds up `bench2b` in particular.

Appendix

Reflection

When implementing the Sudoku solver in CHR we took a glimpse at Thom's implementation. It's given in his book as a solution to some exercise. Once we understood his approach we had the tendency to implement the viewpoints by making use of the same strategy ; constraints representing variables, constraints representing values and a simple implementation of the `first_fail` heuristic. This worked well, but it wasn't the most creative thing to do. Implementing some other approaches (and heuristics) and trying to decrease total runtime was our way to compensate for this.

Workload

Some questions were asked online after all the code had been written. There were four of them, all about Sudoku. One on how to enforce equality of lists (we already had the solution but wanted to be sure there was no better alternative). One on looping through a list which is a subscript of an array (we found the appropriate solution ourselves). One on the inner workings of `ic_global:occurrences/3`. And a final one on memory usage. Aside from a quick experiment with `~=/2` no code was rewritten as a result. None of the questions mentioned Sudoku. All the viewpoints were either thought of by ourselves or come from the literature that was cited in this report.

We started working in mid-April and finished a few days before the deadline, each having worked about ??? hours in total. This includes reading (parts of) the educational material, researching, tinkering, programming, debugging and writing the report.

Overview of the Code

References

[1] Author. Title. *Journal*, 2000.