KU LEUVEN

DEPARTMENT OF COMPUTER SCIENCE

Advanced Programming Languages for A. I.
Final Project

Dooreman–Vandekerkhove

> ⚠ **Attention:** Your report still contains 4 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.)

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

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

Michaël Dooreman
Bruno Vandekerkhove

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 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. They are implemented in the `/sudoku/model` directory. Tests were run on the provided puzzles[2] 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.

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

---

[1] `ECLiPSe` and `CHR` implementations are available in `/sudoku/model/classic.pl` and in `/sudoku/chr/classic.pl`.
[2] `/sudoku/benchmarks/benchmarks.pl` provides automatic benchmarking code.

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.

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 `ic_global: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[3] 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[4] led to an increase in runtime as predicted by Demoen[REF].

» Maybe do some redundant constraints (those two from Simonis) 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.

2 Hashiwokakero

📋 **TODO:** Task 2

3 Scheduling Meetings

📋 **TODO:** Task 3

---

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

[4]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.

A Appendix

A.1 Reflection

> 📄 **TODO:** Invullen kritische reflectie

A.2 Workload

> 📄 **TODO:** Invullen aantal uren gewerkt