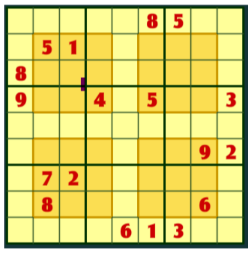
Project 1 (SAT)



The deadline for the SAT project will be 24 November, at noon (Amsterdam time) . This deadline is *hard*: out of fairness to other students, if you hand in after the deadline your mark will be capped to the lowest pass mark. Details of how to hand in can be found under "**Assignments (https://canvas.vu.nl/courses/62832/assignments/230258)** ".

You will have to work in groups of 3 students, please register your group under People/Project Group.

Part I: building a SAT solver

Your challenge is to write a SAT solver, and then use it to solve Sudoku problems. If you are unfamiliar with the rules of a Sudoku puzzle, read up on them here (https://masteringsudoku.com/sudoku-rules-beginners/) , or many other places on the Web.

Writing a SAT solver for Sudoku's requires that you

1. write a SAT solver that can read DIMACS input. See the slides from the third lecture on propositional logic, or here (https://logic.pdmi.ras.ru/~basolver/dimacs.html) ; the "p" line will help you to more easily read your input file.
2. encode the Sudoku rules as clauses in DIMACS format. You get these for free **(https://canvas.vu.nl/courses/62832/pages/project-1-sat-resources) here (https://canvas.vu.nl/courses/62832/pages/project-1-sat-resources) .** (https://www.dropbox.com/s/j3w9d3dzwd3dsbk/sudoku-rules.txt?dl=0)
   Suggestion: make sure you understand what the different lines of this file mean. You will need this later.
3. encode a given puzzle in DIMACS format. Some examples of puzzle in DIMACS is **here as well. (https://canvas.vu.nl/courses/62832/pages/project-1-sat-resources)**
   (again, make sure you understand what the different lines mean).
4. give (2)+(3) as input to (1) and return the solution to the given puzzle.
   This output should again be a DIMACS file, but containing only the truth assignment to all variables (729 for Sudoku, different for other SAT problems). If your input file is called 'filename', then make sure your outputfile is called 'filename.out'. If there is no solution

(inconsistent problem), the output can be an empty file. If there are multiple solutions (eg. non-propert Sudoku) you only need to return a single solution.

Your SAT solver should implement at least three different strategies: the DPLL algorithm without any further heuristics, plus two different heuristics of your choice. These can be some of the heuristics discussed in the lectures, or any other heuristic you can find in the literature (or that you make up yourself, for that matter). Points will be awarded for how sophisticated the strategies are that you choose to implement, but you must implement two different strategies as well as the basic DP algorithm itself.

Of course, your SAT solver must be fully general, in the sense that it is an algorithm to solve not only Sudoku's formulated in SAT, but *any* SAT problem (at least in principle, given enough time and memory).

We give you a **collection of 22000 Sudoku's (https://canvas.vu.nl/courses/62832/pages/project-1-sat-resources)** to test your SAT solver on. These are all in an obvious notation: each line of 81 characters is one puzzle, with a '.' representing an empty square. Obviously, you will need a trivial script to turn such a line into a DIMACS file.

You are free to choose any programming language you fancy, but we must be able to run your SAT solver with the command *SAT -Sn inputfile* , for example: *SAT -S2 sudoku_nr_10 ,* where SAT is the (compulsory) name of your program, n=1 for the basic DP and n=2 or 3 for your two other strategies, and the input file is the concatenation of all required input clauses (in your case: sudoku rules + given puzzle).

Part II: experiment

After you've implemented your SAT solver, we want you to formulate some hypothesis about its behaviour, and then design and execute an experiment to test this hypothesis.

Examples of possible hypotheses would be:

- our strategy 3 outperforms strategy 2, which outperforms basic DP
- Sudoku's with fewer givens are harder, with a linear correspondence between number of givens and runtime
- Sudoku's which are hard for people are also hard for our SAT solver (see here (http://www.sudokuoftheday.com/dailypuzzles/archive/) for an example set, also in other places on the Web)
- We've encoded the rules of a hypersudoku in DIMACS format, and our hypothesis is that solving the same puzzle under hypersudoku rules will be harder than solving it under the normal sudoku rules
- We've encoded the rules of 2x2, 4x4 and 9x9 sudokuo's in DIMACS format, and our hypothesis is that difficulty increases quadratically with the size of the Sudoku.

(there is a rich set of Sudoku variations, see eg here (https://www.sudokudragon.com/sudokuvariants.htm), and many other places on the Web)

(to avoid confusion: the above hypotheses are not necessarily true, they are just examples of the kind of hypotheses we expect).

Clearly, testing these and other hypotheses require that you can somehow measure the performance of your SAT solver. Runtime is of course one measure, but it's also rather uninformative. More informative is to instrument your SAT solver with metrics on the search space it traverses, on the number of backtracks it has to do, on the number of conflicts it encounters, on the number of unit-clauses and pure-literals it resolves, etc. It's up to you to decide which metrics are informative and relevant for your particular hypothesis and experiment.
A note on runtime: we don't care about the *actual* runtime, which will depend on how clever you are as a programmer, which programming language you use, and how much money you spend on a machine. Instead, we only care about the *relative* runtimes of your SAT solver under different experimental conditions.

Points are awarded for the originality of your hypothesis and for the thoroughness of the design and execution of your experiment.

Part III: report

Write a report of max 10 pages where you describe

1. a short introduction to **your research problem**
2. the important **design decisions** for your SAT solver (both how and why)
3. which **heuristics** you implemented (and how and why)
4. your **hypothesis** (plus motivation why it is interesting and plausible)
5. your **experimental design** (which experimental conditions do you test, which test set do you use, which metrics are you measuring, and why)
6. your **experimental results** (consider including plots or graphs or bar charts, and to test for statistical significance of your results)
7. the **conclusion** about your hypothesis that you draw from your results (and why)

Points are awarded for clarity of your argumentation, and quality of the writing and presentation.

- It is compulsory to use Springer style formatting in the style of the Springer Publications format for Lecture Notes in Computer Science (LNCS). For details on the LNCS style, see **Springer's Author Instructions (http://www.springer.com/computer/lncs?SGWID=0-164-6-793341-0)**.
- It is highly recommended that you use Latex for this. With **Overleaf (http://www.overleaf.com)** you have a very nice platform for writing Latex collaboratively. It is worth investing time in this...The easiest is probably to start from **this Overleaf LCNS template. (https://www.overleaf.com/latex/templates/springer-lecture-notes-in-computer-science**

/kzwwpvhwnvfj#.Wm3WNuciG-x)

Here are links to a random selection of 4 good or excellent papers from previous years for inspiration.

- **Paper 1 (https://canvas.vu.nl/courses/62832/files/5387292?wrap=1)** (https://canvas.vu.nl/courses/62832/files/5387292/download?download_frd=1)
- **Paper 2 (https://canvas.vu.nl/courses/62832/files/5387293?wrap=1)** (https://canvas.vu.nl/courses/62832/files/5387293/download?download_frd=1)
- **Paper 3 (https://canvas.vu.nl/courses/62832/files/5387294?wrap=1)** (https://canvas.vu.nl/courses/62832/files/5387294/download?download_frd=1)
- **Paper 4 (https://canvas.vu.nl/courses/62832/files/5387295?wrap=1)** (https://canvas.vu.nl/courses/62832/files/5387295/download?download_frd=1)

What to hand in

You hand in

- all your source code, plus a compiled and executable version of your 'SAT' program (as per the specs above). You do this **here (https://canvas.vu.nl/courses/62832/assignments/230258)**
- your report as a PDF file (this should be anonymous). You submit this file at the Conference system EasyChair at https://easychair.org/conferences/?conf=krvu2022 (https://easychair.org/conferences/?conf=krvu2020)

Please note that these are groups assignment, so please only submit one version of your additional material on Canvas and one paper on EasyChair (please add the names and emails of all authors, though).

Part IV: peer reviewing

You will peer-review two reports from other students. This reviewing will be a task *per person*, so not per team. We will be using the EasyChair conference reviewing system.