CSE 417 25au Homework 8: SAT and NP-completeness

Released: Wednesday, November 26, 2025 @ 11:30am

First due by: Friday, December 5, 2025 @ 11:59pm

Last resubmissions by: Friday, December 5, 2025 @ 11:59pm

Instructions

For Problems 15, 15X.1, and 16, you have four options for submission:

- **Film a video in which you explain your solution.** See the Homework Guide for more details.
- **Use LaTeX to type your solutions.** A template is provided in the "Tasks" page of the course website, if you like.
- **Use Google Docs or Microsoft Word to type your solutions.** If doing so, please use the Equation Editor to ensure that any equations are legible and easy to read.
- **Handwrite your solutions on paper or digitally.** Please write neatly and if on paper, scan in black/white mode, not grayscale.

We prefer either video or LaTeX, but accept any of the 4 options.

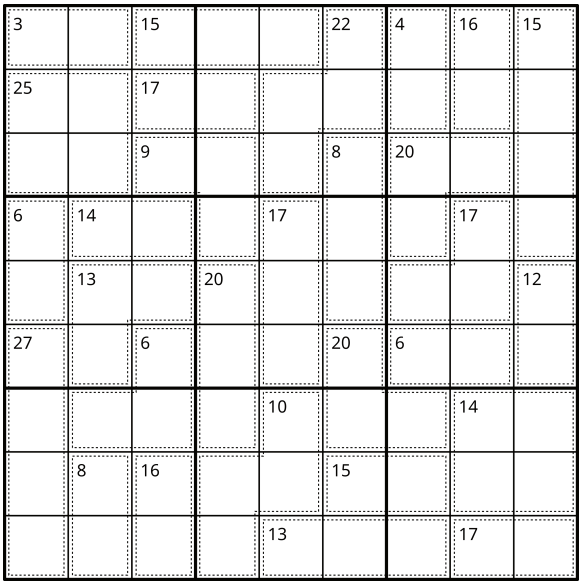
A few more reminders:

- **Submit all problems on Canvas.** Each problem should have its own submission. *Do not* submit one large file containing answers to several problems.
- **Suggested word counts are rough guidelines.** We won't actually count, but if your writing is verbose to the point of obscuring your main argument, we may ask you resubmit more concisely.
- **Review the collaboration policy in the syllabus**. Collaboration is encouraged but strict rules apply, and remember to cite your collaborators.
- If you don't finish in time, we encourage you to be honest and just upload what you have so far. Resubmission won't cost you anything, and we can give you timely feedback on your partial progress by submitting on time.

Happy problem solving!

Problem 15: Killer Sudoku

Sudoku is a common game in which you have to fill a 9×9 grid of numbers such that the digits 1 through 9 each appear exactly once in every row, column, and 3×3 block, with some required starting numbers. But Sudoku is too easy — people across the world have invented more fun variants of Sudoku that are even more challenging! Here is one variant, called Killer Sudoku:



Killer Sudoku follows most of the rules of normal Sudoku, except there are no starting numbers! Instead, the grid is divided into regions with dotted lines. The sum of the numbers in each region is given to you.

In generalized Killer Sudoku, the grid is $N^2 \times N^2$ instead of 9×9, for some appropriate value of N. The grid is broken into r regions, which are given as a list of sums $s_i \in \mathbb{N}$ and regions $R_i \subseteq \{1, \ldots, N^2\}^2$. For example, if the upper-left corner is $(1, 1)$, then the above picture has $s_1 = 3$ and $R_1 = \{(1, 1), (2, 1)\}$. You are required to place numbers $1, \ldots, N^2$ in the cells such that:

- Each row has exactly one each of $\{1, \ldots, N^2\}$.
- Each column has exactly one each of $\{1, \ldots, N^2\}$.
- Each $N \times N$ block has exactly one each of $\{1, \ldots, N^2\}$. Formally, the blocks are $\{aN + 1, \ldots, (a + 1)N\} \times \{bN + 1, \ldots, (b + 1)N\}$ for various values of a and b. (In the picture above, these were drawn with thicker black lines.)
- Each region R_i has sum s_i.

Given an instance of generalized Killer Sudoku, describe a set of variables and a CNF encoding of all the constraints of the problem. You may use any of the advanced building blocks discussed in class without restating how they work, such as a binary adder, etc. For an "E" solution, you should use $\mathsf{poly}(N, r)$ variables and clauses.

(Hint: There are two common ways to represent a number. One way is to use n variables to represent numbers up to n, then use clauses to say that exactly one of these n variables

must be true. The second way is to use $\log(n)$ variables and encode the number in binary. Feel free to use either, or both, for this problem.

If you ever need to convert between the encodings, that's also not too hard. For example, if you have $x_0, x_1, \ldots, x_7$ representing a number up to 7, and you want y_0 to be the least significant bit of the number in binary, you would simply apply a Tseitin transformation to $y_0 \iff (x_1 \vee x_3 \vee x_5 \vee x_7)$. To convert back, for example, if y_0, y_1, and y_2 were the 3 bits of a 3-bit number and you wanted to get x_2, you would Tseitin transform $x_2 \iff (\neg y_0 \wedge y_1 \wedge \neg y_2)$. If you use these Tseitin transformations, you don't have to calculate the converted clauses for this question.)

Solve the problem first, then use the following parts to help structure your answer. These parts are NOT organized in an order that will help you solve the problem.

1. Describe all the variables you will use in your solution. Say in normal English what each variable or group of variables is trying to represent. Then count how many variables you used in big-O, in terms of N and/or r.

   For building blocks such as adders, you do not need to give specific names to the internal variables, but be sure to count them in your total number of variables.

2. Describe all the clauses that you will create in your solution. Say in normal English what each clause or group of clauses is trying to represent. Then count how many clauses you used in big-O, in terms of N and/or r.

   For building blocks such as adders, you do not need to define the clauses in detail (just say, e.g., that there are some clauses corresponding to the Tseitin transformation of an adder), but be sure to count these in your total number of clauses.

Problem 15X: Killer Sudoku (Extensions)

This is an extension problem that doesn't build so much on the ideas of Problem 15, but explores other interesting ideas around Sudoku. **Pick one** of the following to complete:

1. Much of ingenuity involved in both normal Sudoku and Killer Sudoku comes not from solving the puzzle, but from actually writing the puzzle. That's because puzzle writers are subject to rules! First, there must be a *unique* solution to the puzzle. Second, it is also often desirable that the puzzle is *minimal*—removing any clue destroys the uniqueness of the solution. (There are other criteria that are more subjectively measured, like how difficult a puzzle is, whether a puzzle is similarly difficult throughout or spikes in difficulty halfway through solving, etc., but we won't worry about these.)

   For this problem, for simplicity, focus on normal Sudoku (not Killer, but still generalized to $n^2 \times n^2$ grids). To remind you, normal Sudoku does not have regions and sums. Instead, it gives you a set of starting digits in cells that you are required to use as part of your solution. Your goal is to generate random Sudoku puzzles. For this problem, "random" means that every possible valid instance has a non-zero probability of being generated. Assume that you can do things like pick random numbers from a range, randomly permute a list, etc.

   The following two parts will construct a random instance:

   (a) Design an algorithm using at most $\mathsf{poly}(n)$ calls to a SAT solver plus $\mathsf{poly}(n)$ additional time to generate a random *completely filled* Sudoku instance. Briefly justify why every possible valid instance could be generated by your idea.

   (Hint: Guess a random value for a random cell.)

   (b) Given a completely filled Sudoku instance, design an algorithm using at most $\mathsf{poly}(n)$ calls to a SAT solver plus $\mathsf{poly}(n)$ additional time to generate a random minimal Sudoku puzzle whose unique solution is the completely filled instance you started with. Briefly justify why every possible minimal puzzle could be generated by your idea.

2. In this problem, you will gain familiarity with the SAT solver Sat4J in Java. Sat4J is not typically used by people in practice—it is a bit clunky, doesn't support some useful quality-of-life features and extensions of SAT, and is slightly slower by virtue of being written in Java. In real life, it might be easier to use packages like Z3 in languages like Python, but this class is based in Java, and the basic operation of all of these solvers is essentially the same.

   For this problem, for simplicity, focus on normal Sudoku (not Killer, but still generalized to $n^2 \times n^2$ grids). To remind you, normal Sudoku does not have regions and sums. Instead, it gives you a set of starting digits in cells that you are required to use as part of your solution. Your goal is to create a real Sudoku solver in Java. We will teach you the basic syntax of this solver library. Please see the detailed spec online for more details.

   The starter code and spec are available both on the course website and Canvas. Submit your code in the assignment on Canvas (which redirects to Gradescope) or

Gradescope directly, where it will be autograded, with no manual verification. Do not use libraries beyond the standard Java API and Sat4J.

Problem 16: True, False, or Unknown

For each of the statements below, indicate if it is true, false, or unknown. (For example, the statement "$\mathsf{P} = \mathsf{NP}$" would be unknown.) Give a 1–2 sentence explanation for each statement. Both the true/false value and explanation need to be correct for the question to be considered correct. The translation to ESNU will be given by:

E: 6–7 correct with explanations

S: 4–5 correct with explanations

N: 2–3 correct with explanations

U: 0–1 correct with explanations

Here are the questions:

1. There exists a problem in P that is not in NP.
2. There exists a problem in NP that is not in P.
3. There exists a polynomial time algorithm for SAT.
4. If $\mathsf{P} \neq \mathsf{NP}$, then there does not exist a polynomial time algorithm for SAT.
5. Given an undirected graph, there is a way to write down a SAT instance such that the graph has a cycle if and only if the SAT instance is satisfiable.
6. Given a SAT instance, there is a way to write down an undirected graph such that the graph has a cycle if and only if the SAT instance is satisfiable.
7. SAT is NP-hard. For any other problem A, if given an instance of SAT with n variables and m clauses, I can write down an instance of A of size $\mathsf{poly}(n, m)$ that is satisfiable if and only if my instance of SAT was satisfiable, then A is NP-hard.