NAIL062 Project: SAT Solver Application

The following rules, requirements, and deadlines may be modified or supplemented by your instructor.

Your task will be to create a system for solving a given computational problem using a SAT solver. We will solve a *decision problem* (a yes/no answer); if the problem is an optimization problem, solve its decision variant: *Does the given instance have a solution whose objective function has at least/at most the given value?*

Deadlines

- Submission of preferences or proposals for your own problems: **by October 25**
- Submission of the completed project (unless specified otherwise): **by the end of November**

Ethical Rules

You must work completely independently. You are not allowed to consult your solution with anyone except the instructor, nor to use generative AI tools. You may search for supplementary information about the problem (e.g., interesting instances or clarification of the problem statement), but this must not concern its CNF encoding or use of the SAT solver to solve it. You may also not read any code solving this (or related) problem. In case of uncertainty about how to proceed, contact the instructor, who will provide help or guidance.

Assignment Guidelines

Use a GitHub or GitLab repository to develop the project, and commit frequently and regularly so that the progress of your work can be tracked. Use the Python 3 programming language and the SAT solver Glucose 4.2. Do not use any libraries that can convert formulas to CNF or perform similar tasks. Write a script that:

- accepts a problem instance as input (either parameters, a path to a file describing the instance, or a combination of both),
- encodes this instance into SAT in the DIMACS CNF format,
- runs the Glucose SAT solver on the CNF formula,
- decodes the resulting variable assignment (if one exists) back into a solution to the given problem,
- prints this solution in a reasonable, human-readable format,
- optionally prints the constructed CNF formula in DIMACS CNF format and statistics about the SAT solver's progress.

Include several instances with your solution, including at least:

- a small, human-analyzable satisfiable instance,
- a small, human-analyzable unsatisfiable instance (if one exists),

- a satisfiable instance that takes a nontrivial amount of time to solve (at least 10 seconds); if you cannot find one, describe your efforts.

The solution must also include a README.md file with brief documentation containing:

- a precise mathematical description of the problem you are solving (parameters, decision variables, constraints, etc.),
- a description of your chosen CNF encoding and any alternatives,
- user documentation for your script (including a description of the input format and, if it is not obvious, the output format),
- a description of the included instances,
- a description of the experiments you conducted (e.g., how large an instance can your script solve in a reasonable amount of time?), and (if applicable) the code for conducting these experiments.

Example Solution

For inspiration, see the example solution (.zip) for the 15 Puzzle, kindly prepared by Dr. Jirka Švancara (to whom I would like to express my thanks).

Problem Assignment

Each student in a group solves a different problem. You may choose from the list below or propose your own problem of interest (subject to approval; it must be reasonably difficult, so it is preferable to also list preferences from the problems below).

Unless your instructor specifies otherwise, enter your preferences in SIS, in the Study Interim Results module, as comma-separated problem numbers, e.g. `17,3,21,38`. You may list any number of problems in order of preference. You may also email your instructor a proposal for your own problem, including a link to a detailed definition and an explanation of why it interests you.

Submission and Presentation

Submit the completed project in the SIS module Study Group Roster (unless your instructor specifies otherwise), as a link to a GitHub or GitLab repository. The repository may be public, or you may grant your instructor access. The project's development history must be apparent from the repository; uploading only the final version of the script or documentation is not acceptable.

As part of the submission, you may be asked to give a short presentation of the project, demonstrate that the code works, and describe your solution and implementation decisions; your instructor will provide further details.

List of Problems

1. Betweenness Problem
2. Bipartite Dimension
3. Cubic Subgraph (problem no. 26)
4. Degree-constrained Spanning Tree
5. Domatic Number
6. Feedback Arc Set

7. Futoshiki
8. Garden of Eden
9. Graph Isomorphism
10. Graph Partitioning (problem no. 25)
11. Hamiltonian Path
12. Hidato
13. Hitori
14. Induced Subgraph Isomorphism
15. Intersection Number
16. Killer Sudoku, generalized to an $n \times n$ grid (where n is a perfect square)
17. Largest Common Subgraph (problem no. 84)
18. Longest Circuit (problem no. 17)
19. Maximum Density Still Life
20. Multi-agent pathfinding, consider only grid graphs, makespan objective
21. Nonogram
22. Numberlink
23. Nurikabe, restricted to clues 1 and 2
24. Peaceably Co-existing Armies of Queens, generalized to an $n \times n$ grid
25. Rural Postman (problem no. 34)
26. Shortest Common Superstring (problem no. 16)
27. Slitherink
28. Social Golfers Problem
29. Sports Tournament Scheduling
30. Square-Tiling (problem no. 54)
31. Stable Roommates
32. Steiner Triple Systems
33. Steiner Tree, consider only unit edge weights
34. Subgraph Isomorphism
35. Tetris
36. Word Design for DNA Computing on Surfaces