MiniPyDPLL: A Python Implementation of the DPLL Algorithm Inspired by MiniSAT

Abstract. MiniSAT, a renowned CDCL solver developed in C++, is lauded for its minimalist style, making it a great educational tool for efficient CDCL solver development. However, its comprehensive feature set, strict input parsing rules, extensive statistics reporting, and advanced techniques can make it challenging for beginners to grasp. In contrast, PyMiniDPLL is a simplified, Python-based implementation of the DPLL algorithm, inspired by MiniSAT's source code. It aims to serve as an educational tool for students and newcomers to the field of SAT solvers. This implementation excludes non-essential components for DPLL, thereby reducing complexity and focusing on core concepts like watched literals and variable heaps. The initial version assumes well-formed CNF inputs, prioritizing code clarity and instructional value over robustness. PyMiniDPLL serves as a stepping stone towards understanding and developing efficient CDCL solvers, providing a smoother learning curve compared to more complex solvers like MiniSAT. Future work includes creating detailed tutorials and potentially extending the implementation to cover CDCL solvers, offering a comprehensive educational pathway from DPLL to CDCL.

Report

MiniSAT [1] was developed in C++ with a minimalist style to reduce the learning curve for writing efficient CDCL solvers. It has achieved this goal successfully, as evidenced by the many subsequent solvers inspired by its source code. However, in striving to be full-featured and user-friendly, MiniSAT includes many detailed features such as strict input formula parsing rules, extensive statistics reporting, and advanced techniques beyond the core components of a CDCL solver. This can make learning from its source code somewhat time-consuming, especially for those new to the field or learning independently.

Experienced developers or those with a solid background in CDCL solvers might navigate these complexities comfortably. However, based on my experience—learning these concepts through self-study and offline materials—this is not the most accessible approach. For instance, while the learning component of the CDCL solver is well-documented, the implementation of watched literals is less transparent. Although tutorials and textbooks explain watched literals clearly, effectively implementing them remains challenging.

With this in mind, I aimed to create a more approachable codebase for students rather than professionals interested in implementing CDCL solvers. I translated and slightly modified the DPLL algorithm into Python, borrowing from MiniSAT's source code. I chose to implement DPLL instead of CDCL for two main reasons. First, DPLL is still used in research [2], so understanding how to convert a CDCL codebase to DPLL can be useful. Second, learning complex concepts is often easier through multiple passes rather than all at once. Given that conflict learning in CDCL solvers is well-documented and other non-DPLL components of CDCL solvers, e.g., ‘restart’ are relatively easy to understand, starting with DPLL seemed beneficial.

This approach simplifies the code by reducing the number of variables and functions compared to the original MiniSAT, making it easier for first-time readers to grasp the relationships between different code components. The code retains most of the main components of a CDCL solver, such as watched literals and variable heaps, and simplifies data types like clauses and literals, at the expense of not fully leveraging modern CPU architecture.

In the current version of the code, I assumed that the input CNF files are valid and clean, meaning I did not implement checks for basic things like the presence of a literal and its complement in the same clause. While this may cause the code to fail in certain situations, it works as intended when the input file meets these assumptions, as demonstrated in the tests. This decision was made to keep the code appropriate for instructional purposes. Once learners are comfortable with this code, they should be able to understand and modify MiniSAT's or other CDCL solvers' original source codes for their purposes, such as exploring new ideas and features.

Although I have only written the code so far, creating a text or video tutorial to explain it would make it more appealing. Additionally, I might consider to write a CDCL solver and develop two consecutive tutorials: one that teaches how to write a DPLL solver and another that extends it to a CDCL solver.

References:

[1] Eén, Niklas, and Niklas Sörensson. "An extensible SAT-solver." International conference on theory and applications of satisfiability testing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[2] Cameron, C., Hartford, J., Lundy, T., Truong, T., Milligan, A., Chen, R. and Leyton-Brown, K., 2022. Monte Carlo Forest Search: UNSAT Solver Synthesis via Reinforcement learning. arXiv preprint (to appear November 21st).