Comparative Analysis of Boolean Satisfiability Algorithms

TEAM NAME - Fab Five

Introduction & Project Objectives

The Boolean Satisfiability Problem (SAT) is one of the most important puzzles in computer science. It's the classic example of an NP-complete problem—meaning it's hard to solve efficiently, yet incredibly powerful. Figuring out SAT quickly isn't just a theoretical exercise; it has real-world impact. It's used in hardware and software verification, automated testing, planning, scheduling, and many other areas where we need to be absolutely certain that systems work the way they're supposed to.

Implement Diverse Solvers

Develop both complete (DPLL, CDCL) and incomplete (WalkSAT, probSAT) SAT solvers from scratch.

Analyze Performance

Evaluate solver efficiency across various benchmarks: Random 3-SAT instances (especially at the phase transition) and structured problems like Sudoku.

Evaluate Modern Heuristics

Assess the practical impact of advanced techniques such as VSIDS (Variable State Independent Decaying Sum) and Clause Learning on solver performance.

Methodology: Benchmarks & Test Harness

Benchmark Generation

- **Random 3-SAT Instances:** Generated at the theoretical phase transition ratio of 4.26 clauses per variable.
  - **Sizes:** 50, 100, 150, 200 variables.
  - **Goal:** To meticulously test scalability and observe behavior around the SAT/UNSAT phase transition.
- **Structured Sudoku Problems:** Utilized a custom `sudoku_encoder.py` to translate standard 9x9 Sudoku puzzles into Conjunctive Normal Form (CNF).
  - **Configuration:** 729 variables, approximately 12,000 clauses.
  - **Variety:** Included both satisfiable and unsatisfiable Sudoku instances to test solver robustness.

Automated Test Harness

- **Script:** An automated Python script (`run_experiments.py`) orchestrated the execution of all implemented solvers across the benchmarks.
- **Key Metrics Captured:**
  - CPU Time (seconds)
  - Peak Memory Usage (MB)
  - Number of Decisions
  - Number of Flips (for SLS solvers)
  - Number of Conflicts

DPLL Variants: The Foundation of Complete Solvers

Baseline DPLL

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm forms the backbone of many modern SAT solvers. It employs a recursive backtracking search combined with crucial simplification rules.

- **Core Logic:** Backtracking search.
- **Optimizations:** Unit Propagation and Pure Literal Elimination significantly prune the search space.
- **Complexity:** Remains O(2^n) in the worst case.

DPLL + Jeroslow-Wang (JW-TS)

The Jeroslow-Wang Two-Sided (JW-TS) heuristic enhances DPLL by intelligently selecting decision variables, aiming to trigger early propagations.

- **Heuristic:** Prioritizes literals that appear in a large number of short clauses, maximizing potential unit propagations.
- **Score:** Calculated as $J(\ell) = \sum_{(\ell \in c)} 2^{\{-|c|\}}$, where c is a clause containing literal ℓ and $|c|$ is its length.
- **Impact:** This strategic choice often leads to a significant reduction in the effective search tree depth.

CDCL: Learning from Conflicts

The Concept: "Learning from Mistakes"

Conflict-Driven Clause Learning (CDCL) is a major leap forward in how SAT solvers work. Instead of blindly exploring possibilities, CDCL learns from its mistakes. When the solver hits a conflict, it analyzes what went wrong and creates a new clause that prevents it from getting stuck in the same situation again. This “learning from conflicts” approach dramatically cuts down the search space and makes modern SAT solvers far more efficient.

01

1-UIP Learning

Analyzes the implication graph to identify the "first unique implication point" (1-UIP), which represents the most general reason for a conflict. This clause is then added to the clause database.

02

Non-Chronological Backjumping

Instead of simply backtracking to the most recent decision, CDCL jumps back multiple levels to the decision point directly responsible for the learned conflict, efficiently cutting off irrelevant branches.

03

VSIDS Heuristic

The Variable State Independent Decaying Sum (VSIDS) heuristic dynamically prioritizes variables that have been recently involved in conflicts, promoting earlier detection of critical variables.

04

2-Watched Literals

A lazy data structure optimization that dramatically improves the efficiency of unit propagation by only checking clauses when one of their two watched literals becomes false.

Stochastic Local Search (SLS) Solvers

WalkSAT: Escaping Local Minima

- **Logic:** Iteratively attempts to satisfy clauses by flipping variables within unsatisfied clauses.
- Noise (**p**): A critical parameter; with probability p, a random variable in an unsatisfied clause is flipped. Otherwise, a "greedy" flip is chosen that maximizes the number of satisfied clauses.
- **Strength:** The introduction of noise allows WalkSAT to escape local minima in the search landscape, enabling it to explore diverse parts of the solution space.

probSAT: Fully Probabilistic Approach

- **Logic:** Unlike WalkSAT's greedy-with-noise approach, probSAT employs a fully probabilistic selection mechanism.
- **Heuristic:** Variables are selected for flipping based on a probability distribution derived from their "break scores," which are often calculated using a polynomial function.
- **Advantage:** By avoiding a hard greedy step, probSAT can be more robust in certain problem instances and less prone to getting stuck in highly localized optima.

Key Implementation Challenges

1

CDCL Complexity

Implementing the implication graph and ensuring correct backtracking levels was particularly challenging. Resetting `level_literals` during non-chronological backtracking required meticulous attention to detail to avoid subtle bugs.

2

Sudoku Encoding

Mapping the intricate 9x9 Sudoku grid constraints (row, column, and 3x3 box uniqueness) into CNF variables demanded precise index management and careful formulation of clauses to accurately represent the puzzle.

3

Parameter Tuning

Determining the optimal noise parameter (p) for WalkSAT required extensive experimentation. A comprehensive parameter sweep was necessary to identify the sweet spot for maximum success rates across various problem types.

Results: Scalability on Random 3-SAT

Our experiments with random 3-SAT instances revealed critical insights into solver scalability, particularly as problem size increased.

1

Small Instances (50 Variables)

All DPLL variants achieved **100% success** within a short timeframe (~0.10s - 0.27s). CDCL, however, struggled (25% success), indicating that its overhead outweighs benefits on smaller, easier problems.

2

The "Wall" (150+ Variables)

A significant scalability barrier was observed at 150+ variables. All complete solvers timed out on 200-variable instances (0% success). DPLL-JW managed 40% on 150 variables but failed to scale further.

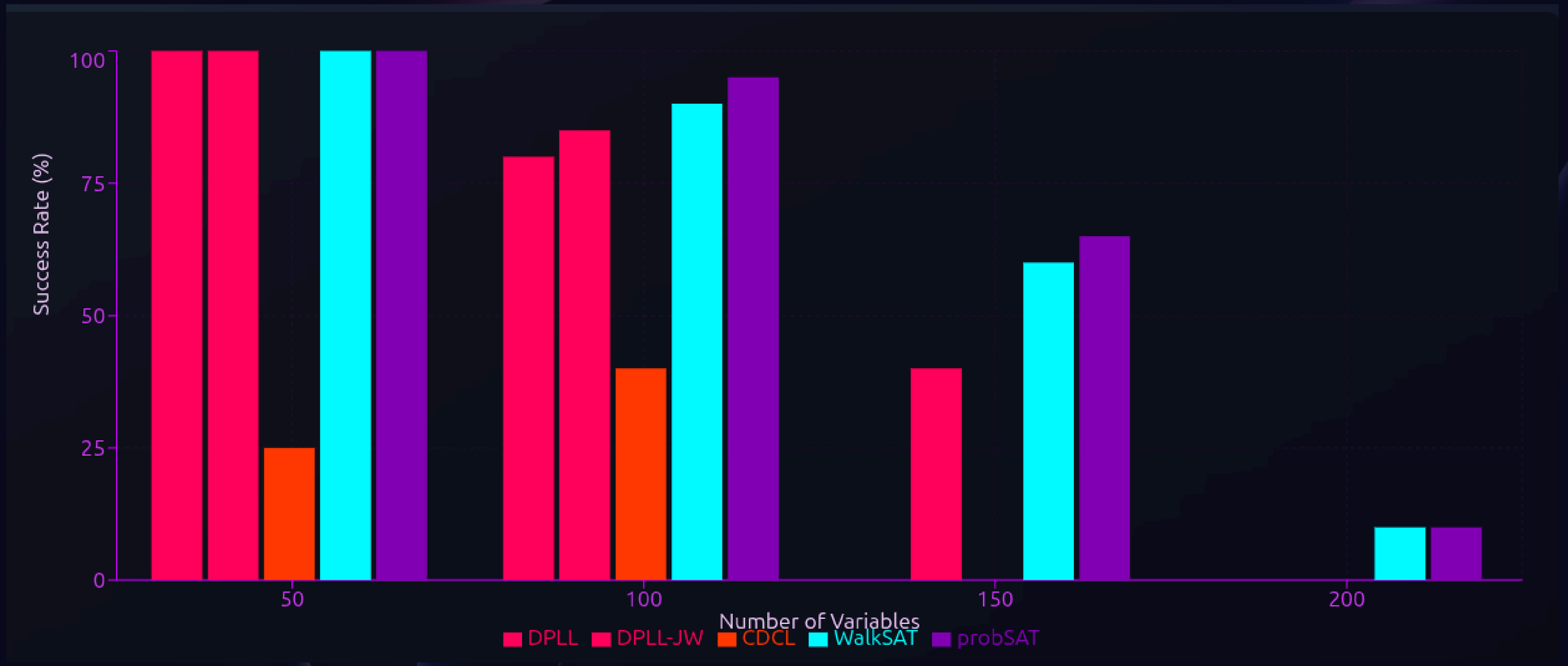
3

SLS Dominance

WalkSAT and probSAT were the **only** solvers capable of handling 200-variable instances. While their success rate was 10% (on hard instances), they solved problems in approximately ~4.5s when successful.

Results: Scalability on Random 3-SAT

Results: Scalability on Random 3-SAT



Results: Robustness on Structured Sudoku

Structured problems like Sudoku present a different challenge, where the inherent constraints can either aid or hinder various SAT algorithms.

CDCL's Structured Dominance

CDCL emerged as the clear winner for structured problems, achieving an average solution time of **0.16s with 100% success**. This highlights CDCL's ability to "short-circuit" the search by learning from specific conflict patterns inherent in structured CNF.

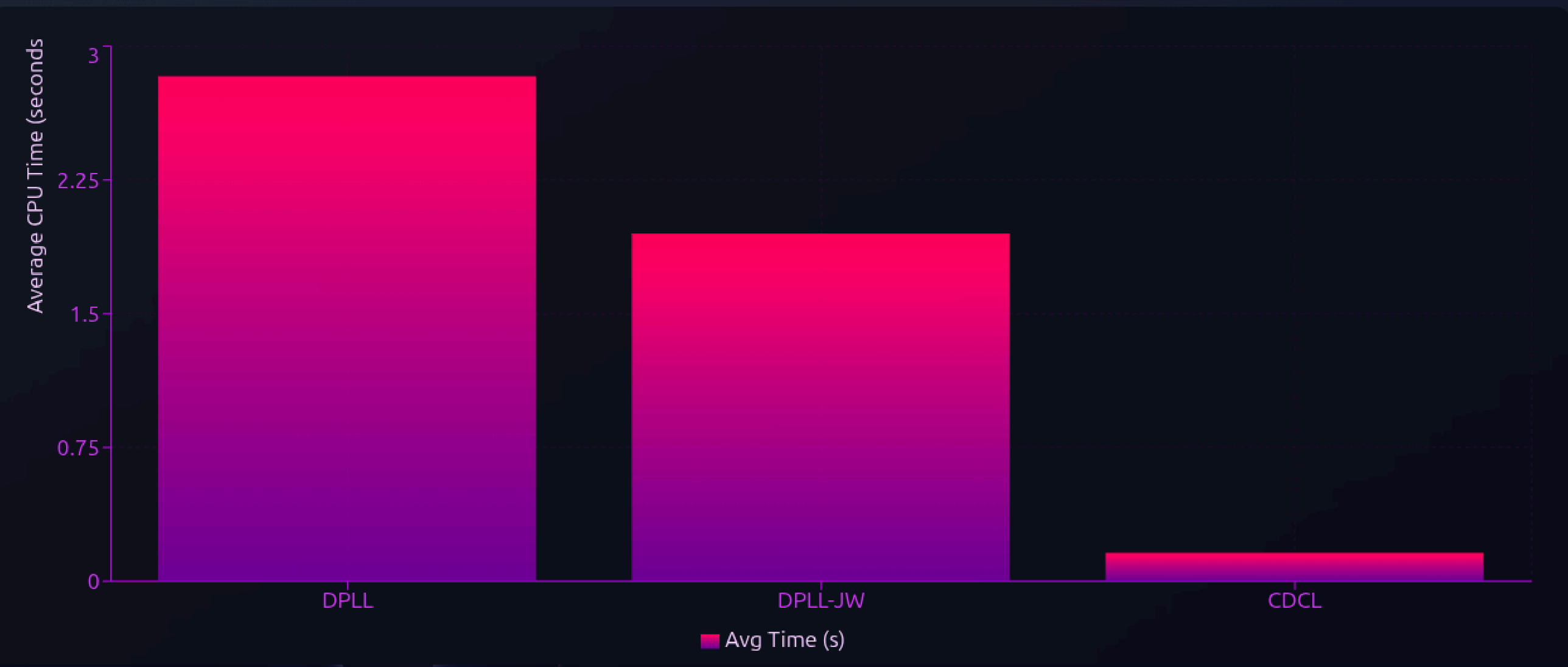
DPLL Performance

The baseline DPLL, while complete, took significantly longer, with an average time of 2.83s. This indicates that without advanced learning mechanisms, the search space for Sudoku problems remains vast and complex.

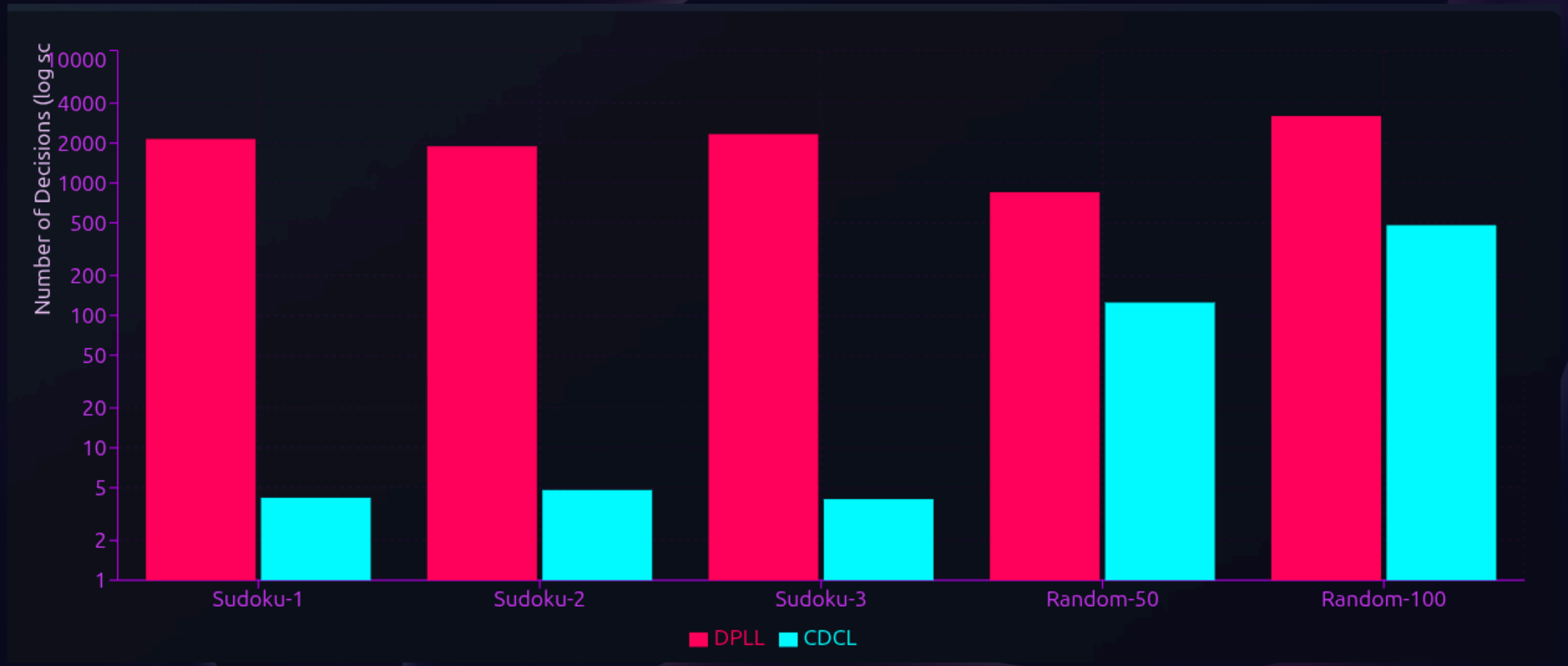
SLS Failure

Stochastic Local Search (SLS) algorithms, WalkSAT and probSAT, recorded **0% success** on the Sudoku instances. This suggests that local search methods struggle to navigate the highly constrained, "narrow canyons" of the Sudoku solution space, where small random flips rarely lead to a solution.

Results: Robustness on Structured Sudoku

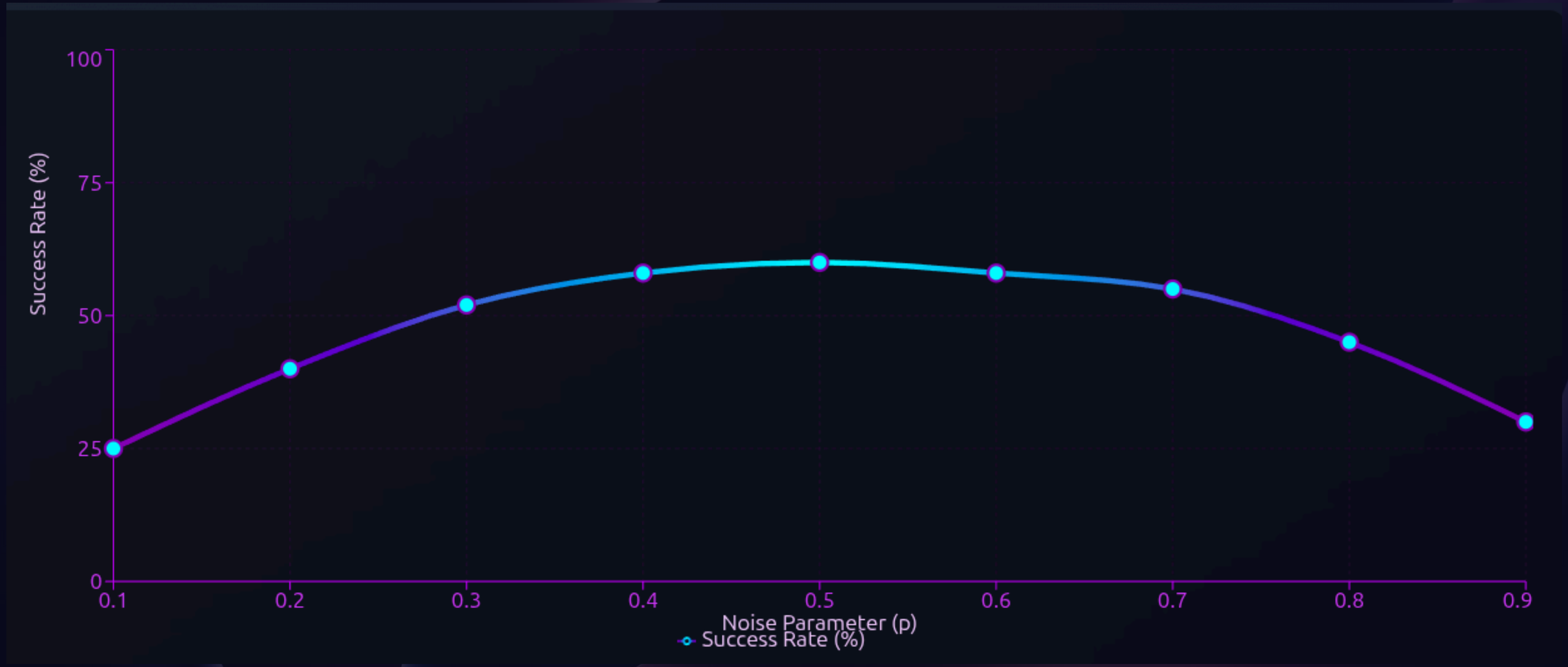


Results: Heuristics & Sensitivity Analysis



Decision Reduction

Results: Heuristics & Sensitivity Analysis



WalkSAT Sensitivity to Noise

Bonus Achievements

We extended our project beyond the core requirements, delivering several key enhancements that deepened our understanding and refined our implementations.

Full CDCL Implementation

Integrated advanced features like 1-UIP conflict analysis and non-chronological backjumping for a robust solver.

probSAT Solver

Developed a second Stochastic Local Search algorithm, enabling a more comprehensive comparison of incomplete approaches.

Parameter Sensitivity

Conducted a rigorous analysis of WalkSAT's noise parameter to understand its impact on solver performance.

Sudoku Encoder

Created a custom Python script to convert Sudoku puzzles into Conjunctive Normal Form (CNF), expanding our benchmark capabilities.

Conclusion & Future Work

Summary of Key Insights

- **CDCL:** Emerges as the most effective strategy for structured problems like Sudoku, owing to its conflict learning and non-chronological backjumping capabilities.
- **SLS (Stochastic Local Search):** Proves indispensable for tackling large, satisfiable random instances where complete solvers struggle to scale.
- **Heuristics:** Modern heuristics are not merely optimizations but fundamental components that dramatically enhance solver scalability and efficiency across problem types.

Future Work

Our next steps involve integrating advanced restart strategies into the CDCL solver to mitigate performance plateaus and optimizing Python data structures for further efficiency gains across all implementations.

THANK YOU :)