Institut für Informatik
der Universität München
Dr. J. Johannsen
Florian Leimgruber

WS 2020/21

21. January 2021

Exercises
SAT Solving

Problem set 5

Project 3: Extending your DPLL-Solver

The goal of this project is to extend your DPLL-Solver from project 2 to a CDCL-Solver. The rules regarding languages and team size are therefore the same as in project 2. There are four parts to this project:

1. Compulsory part: Implement CDCL. This includes:
    - Deriving an asserting conflict clause on every conflict, and adding it to the clause database. The learning schema is up to you.
    - Non-chronological backtracking, based on the assertion level of the learned clause.
    - A clause deletion strategy.
    - The VSIDS or VMTF branching heuristic.
    - Watched literals.
2. Bonus part: Implement restarts. Experiment with phase saving and different restart policies.
3. Bonus part: Implement some preprocessing techniques.
4. Bonus part: Implement proof logging in DRUP (or DRAT) format, verifiable with DRAT-trim. Add a command line option to your solver to enable it, and write the proof to a file. Make sure to justify any preprocessing in your proofs, or disable it when generating them. Proof logging will be covered in the lecture soon.

Your solver is supposed to solve the inputs provided with project 2 within a time limit of 1 minute/input (all of them, not just the ones in test anymore). Benchmark your solver and document the performance gains of your experiments. Generate cactus plots.

Optionally, benchmark your solver against inputs used at the annual SAT competition. We recommend using inputs of older competitions, such as 2006 to

2010, because the time and memory limits (and difficulty of the problems) in more recent years are pretty high.

Solutions can be submitted via uni2work until 15. March, 14:00. Please add a short readme file, explaining how to build/run your program.

Exercise 16: Recall the random walk algorithm from the lecture:

```
pick $\alpha$ randomly
repeat $2n^2$ times
  if $\alpha \models F$
    then return $\alpha$
  pick $C = a_1 \vee \ldots \vee a_k$ with $C\alpha = 0$
  pick $i$ randomly from $\{1, \ldots, k\}$
  $\alpha := \alpha$ with $[a_i \leftarrow 1]$
return UNSAT
```

Contrary to the lecture, this version uses $2n^2$ iterations. Show that this algorithms solves 2-SAT with a one-sided error probability of at most $\frac{1}{2}$.

Exercise 17: The problem MAX-E3SAT is defined as follows: *Given a E3-SAT instance, find an assignment that maximises the number of satisfied clauses.* Show that:

1. MAX-E3SAT is NP-hard.
2. For any E3-SAT instance, there exists an assignment that satisfies at least $\frac{7}{8}$ of it's clauses.
3. Design a randomised algorithm that finds such an assignment with probability at least $\frac{1}{2}$.
4. Any E3-SAT instance with at most 7 clauses is satisfiable.

Note: Similar results are known for general MAX-3SAT (paper).

Exercise 18: Recall the Set-Cover problem, which was used to derandomize the Hamming ball algorithm: Given a set S and a family of sets $F \subseteq S$, find $C \subseteq F$ such that $S = \bigcup C$. The lecture relied on the following greedy algorithm to find a $\log |S|$ approximation of the minimal set cover:

```
$C := \emptyset$
while $S \not\subseteq \bigcup C$
  pick $c \in F$ s.t. $c \setminus \bigcup C$ is maximal
  $C := C \cup \{c\}$
```

Proof that this algorithm indeed finds a $\log |S|$ approximation.

Bonus: There is a neat variation of the Hamming ball algorithm solving general SAT in $\mathcal{O}(2^{n-0.712\sqrt{n}})$, independent of the maximum clause size. The key idea is picking small clauses while exploring the hamming balls, similar to how we optimised DPLL-style algorithms earlier. See paper.