Morally correct way of solving SAT

Angry goats

October 2024

SAT problem

Boolean Satisfiability Problem is one of the most important and fundamental problem in Computer Science

- Does exist a variable combination (True, False values), that satisfies a given logical formula?
- Formula contains more than one different variable and more than one binary logical operator.
- 3-SAT instance:

$$(X \vee X \vee Y) \wedge (\neg X \vee \neg Y \vee \neg Y) \wedge (\neg X \vee Y \vee Y)$$

History

The history of SAT started in the 19. century.

- In the early 1900s, the works of Alan Turing and Kurt Gödel helped the formalization of problems like SAT.
- In 1971 Stephen Cook published, that the SAT is and NP-complete problem.
- In 1980s the demand of SAT solver efficient algorithms has increased.
- 60s most famous SAT solver: DPLL

Famous approaches

Main solving methods through the history.

- DPPL algorithm searches for solutions systematically.
- VSIDS algorithm use heuristic methods to find solutions.
- CDCL tries to learn from conflicts during execution.
- The 90s Stochastic Local search works efficiently with practical SAT eases.

Fields of applications

Main solving methods through the history.

- Logical electric circuits and the programs formal control.
- Code optimization task.
- Cryptography algorithms and security analysis.

Related works

- Parallel SAT with single- and multicore CPU-s
- Problem with too many Cores and Memory
- Principles for a great Parallel SAT

Background

Boolean formulas and satisfiability

- Propositional variable
- Interpretation
- Formula
  1. propositional variable
  2. $\neg X$
  3. $(X \circ Y)$, where $\circ$ can be $\wedge$, $\vee$, $\supset$

  Every formula is created by using the previous rules finitely many times.
- Interpretation satisfies a formula
- Satisfiable formula

Background

Conjunctive normal form

- Literals
- Clause
- Conjunctive normal form (CNF)

Methodology

- Combine the innovations of MiniSAT and ManySAT into one parallel SAT solving algorithm.
- ManySAT’s cooperative search strategy with MiniSat’s conflict-clause minimization techniques.
- Quicker convergence towards solutions

For Proof of convergence and and correctness see the paper.

Methodology2

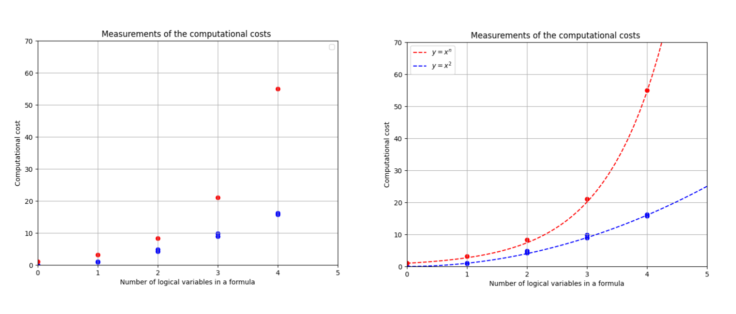
Data: D : Set of clasuses
Result: S : Bool
$\forall i, S_i \leftarrow$ *MiniSat_Init* ;
$D_i \leftarrow$ *PartitionD*;
$S \leftarrow$ *false*; $d \leftarrow 1$ *confPool* $\leftarrow$ *ConflictPool$_I$nit*;
while $\neg S$ **do**
  Solve D_i with each S_i, with depth d;
  if S_i *found conflicting clauses* **then**
    Add the conflict to *confpool*;
    $d \leftarrow d + f(\text{confPool.numOfClauses})$;
  else
    Partition each D_i further;
  end
end

Algorithm 1: Clause sharing

The $f(x)$ function is freely choosable, but testing has shows that the best result come from *log*.

Measurements

The team tested the best available algorithms, as well as the one developed by the team.



Discussion

- Runtime after the optimalization
- Hardware Cost decrase
- A* algorithm

Conclusion

- Computational Cost
- MiniSAT and ManySAT
- Concurrent Algorithm

Future work

Goal

- faster algorithm
- optimize the computational costs and the runtime
- from $\theta(n^2)$ to $\theta(n * log(n))$