Foundations of Artificial Intelligence

32. Propositional Logic: Local Search and Outlook

Thomas Keller and Florian Pommerening

University of Basel

April 26, 2023

Foundations of Artificial Intelligence
April 26, 2023 — 32. Propositional Logic: Local Search and Outlook

32.1 Local Search: GSAT

32.2 Local Search: Walksat

32.3 How Difficult Is SAT?

32.4 Outlook

32.5 Summary

Propositional Logic: Overview

Chapter overview: propositional logic
- ▶ 29. Basics
- ▶ 30. Reasoning and Resolution
- ▶ 31. DPLL Algorithm
- ▶ 32. Local Search and Outlook

32.1 Local Search: GSAT

Local Search for SAT

- ▶ Apart from systematic search, there are also successful local search methods for SAT.
- ▶ These are usually not complete and in particular cannot prove unsatisfiability for a formula.
- ▶ They are often still interesting because they can find models for hard problems.
- ▶ However, all in all, DPLL-based methods have been more successful in recent years.

Local Search for SAT: Ideas

local search methods directly applicable to SAT:

- ▶ candidates: (complete) assignments
- ▶ solutions: satisfying assignments
- ▶ search neighborhood: change assignment of one variable
- ▶ heuristic: depends on algorithm; e.g., #unsatisfied clauses

---

GSAT (Greedy SAT): Pseudo-Code

auxiliary functions:

- ▶ violated(Δ, I): number of clauses in Δ not satisfied by I
- ▶ flip(I, v): assignment that results from I when changing the valuation of proposition v

```
function GSAT(Δ):
repeat max-tries times:
    I := a random assignment
    repeat max-flips times:
        if I ⊨ Δ:
            return I
        V_greedy := the set of variables v occurring in Δ
                    for which violated(Δ, flip(I, v)) is minimal
        randomly select v ∈ V_greedy
        I := flip(I, v)
return no solution found
```
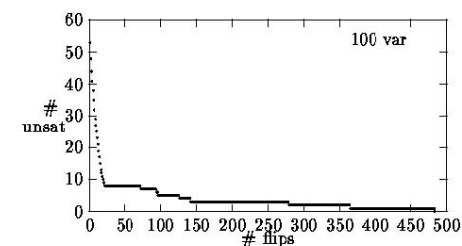

GSAT: Discussion

GSAT has the usual ingredients of local search methods:

- ▶ hill climbing
- ▶ randomness (although relatively little!)
- ▶ restarts

empirically, much time is spent on plateaus:

32.2 Local Search: Walksat

Walksat: Pseudo-Code

$lost(\Delta, I, v)$: #clauses in Δ satisfied by I, but not by $flip(I, v)$

function Walksat(Δ):
repeat *max-tries* **times**:
    $I :=$ a random assignment
    repeat *max-flips* **times**:
        if $I \models \Delta$:
            return I
        $C :=$ randomly chosen unsatisfied clause in Δ
        if there is a variable v in C with $lost(\Delta, I, v) = 0$:
            $V_{choices} :=$ all such variables in C
        else with probability p_{noise}:
            $V_{choices} :=$ all variables occurring in C
        else:
            $V_{choices} :=$ variables v in C that minimize $lost(\Delta, I, v)$
        randomly select $v \in V_{choices}$
        $I := flip(I, v)$
return no solution found

Walksat vs. GSAT

Comparison GSAT vs. Walksat:

► much more randomness in Walksat
because of random choice of considered clause

► "counter-intuitive" steps that temporarily increase
the number of unsatisfied clauses are possible in Walksat

⤳ smaller risk of getting stuck in local minima

32.3 How Difficult Is SAT?

How Difficult is SAT in Practice?

- ▶ SAT is NP-complete.
- ↝ known algorithms like DPLL
  need exponential time in the worst case
- ▶ What about the average case?
- ▶ depends on how the average is computed
  (no "obvious" way to define the average)

SAT: Polynomial Average Runtime

Good News (Goldberg 1979)
construct random CNF formulas
with n variables and k clauses as follows:
In every clause, every variable occurs
- ▶ positively with probability $\frac{1}{3}$,
- ▶ negatively with probability $\frac{1}{3}$,
- ▶ not at all with probability $\frac{1}{3}$.
Then the runtime of DPLL in the average case
is polynomial in n and k.

↝ not a realistic model for practically relevant CNF formulas
(because almost all of the random formulas are satisfiable)

Phase Transitions

How to find interesting random problems?

conjecture of Cheeseman et al.:

Cheeseman et al., IJCAI 1991
Every NP-complete problem has at least one size parameter
such that the difficult instances are close to a critical value
of this parameter.

This so-called phase transition separates two problem regions,
e.g., an over-constrained and an under-constrained region.

↝ confirmed for, e.g., graph coloring, Hamiltonian paths and SAT

Phase Transitions for 3-SAT
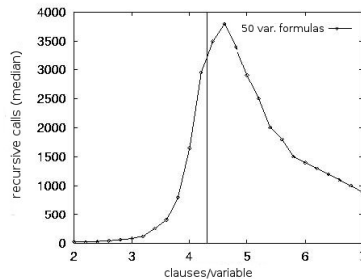
Problem Model of Mitchell et al., AAAI 1992
- ▶ fixed clause size of 3
- ▶ in every clause, choose the variables randomly
- ▶ literals positive or negative with equal probability

critical parameter: #clauses divided by #variables
phase transition at ratio ≈ 4.3
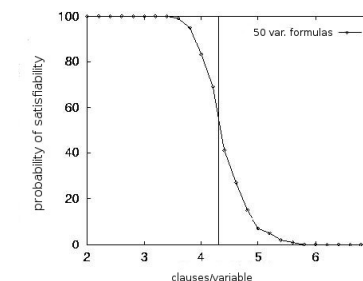
Phase Transition of DPLL

DPLL shows high runtime close to the phase transition region:

Phase Transition: Intuitive Explanation

▶ If there are many clauses and hence the instance is unsatisfiable with high probability, this can be shown efficiently with unit propagation.

▶ If there are few clauses, there are many satisfying assignments, and it is easy to find one of them.

▶ Close to the phase transition, there are many "almost-solutions" that have to be considered by the search algorithm.

32.4 Outlook

State of the Art

▶ research on SAT in general:
⤳ http://www.satlive.org/

▶ conferences on SAT since 1996 (annually since 2000)
⤳ http://www.satisfiability.org/

▶ competitions for SAT algorithms since 1992
⤳ http://www.satcompetition.org/
  ▶ largest instances have more than 1 000 000 literals
  ▶ different tracks (e.g., SAT vs. SAT+UNSAT; industrial vs. random instances)

More Advanced Topics

DPLL-based SAT algorithms:
- ▶ efficient implementation techniques
- ▶ accurate variable orders
- ▶ clause learning

local search algorithms:
- ▶ efficient implementation techniques
- ▶ adaptive search methods ("difficult" clauses
  are recognized after some time, and then prioritized)

SAT modulo theories:
- ▶ extension with background theories
  (e.g., real numbers, data structures, . . .)

32.5 Summary

Summary (1)

- ▶ local search for SAT searches in the space of interpretations;
  neighbors: assignments that differ only in one variable
- ▶ has typical properties of local search methods:
  evaluation functions, randomization, restarts
- ▶ example: GSAT (Greedy SAT)
  - ▶ hill climbing with heuristic function: #unsatisfied clauses
  - ▶ randomization through tie-breaking and restarts
- ▶ example: Walksat
  - ▶ focuses on randomly selected unsatisfied clauses
  - ▶ does not follow the heuristic always, but also injects noise
  - ▶ consequence: more randomization as GSAT
    and lower risk of getting stuck in local minima

Summary (2)

- ▶ more detailed analysis of SAT shows: the problem
  is NP-complete, but not all instances are difficult
- ▶ randomly generated SAT instances are
  easy to satisfy if they contain few clauses, and
  easy to prove unsatisfiable if they contain many clauses
- ▶ in between: phase transition