Propositional Satisfiability

~~and Constraint Satisfaction~~

(preliminary slide set based on a presentation by Suling Yang)

The SAT Problem

- Given a propositional formula F, decide whether there exists an assignment a of truth values to the variables in F such that F is true under a.

- SAT algorithms are typically restricted to CNF formulae as input; these arise naturally in many applications of SAT (in other cases, CNF transformations are used)

Polynomial Simplifications

- Elimination of duplicate literals and clauses:
  - E.g. $(a \vee b \vee a) \wedge (a \vee b) = (a \vee b) \wedge (a \vee b) = (a \vee b)$
- Elimination of tautological clauses:
  - E.g. $(a \vee \neg a) = T$
- Elimination of subsumed clauses:
  - E.g. $(a \vee b) \wedge (a \vee b \vee c) = (a \vee b)$
- Elimination of clauses containing pure literals

Unit Propagation

- *Unit clause*: a clause consisting of only a single literal.
  - E.g. $(a) \vee (\neg a \vee b)$

- *Unit Resolution*:
  - E.g. $(a) \vee (\neg a \vee b) = (b)$

- *Complete unit propagation*: repeat application of unit resolution until:
  - no more unit clause, or
  - empty clause, or
  - no more clauses.

Practical Applications of SAT

- Hardware verification:
  Bounded Model Checking (BMC)

- Asynchronous circuit design:
  Complete State Coding (CSC) Problem in State Transition Graphs (STGs)

- Sports scheduling problems:
  Finding fair schedules for basket ball tournaments

Generalisations and Related Problems

- Constraint Satisfaction Problems, in particular:
  - Multi-Valued SAT (MVSAT)
  - Pseudo-Boolean CSPs
- MAX-SAT (unweighted and weighted)
- Dynamic SAT (DynSAT)
- Propositional Validity Problem (VAL)
- Satisfiability of Quantified Boolean Formulae (QSAT)
- #SAT

The GSAT Architecture

- Based on 1-exchange neighbourhood

- Evaluation function $g(F,a)$ maps each variable assignment a to the number of clauses of the given formula F unsatisfied under a (note: $g(F,m)=0$ iff m is a model of F)

- GSAT algorithms differ primarily in the method used for selecting the variable to be flipped in each step

- Initialisation: Random picking from space of all variable assignments.

The Basic GSAT Algorithm

procedure *GSAT(F, maxTries, maxSteps)*

  input: CNF formula *F*, positive integers *maxTries* and *maxSteps*

  output: model of *F* **or** 'no solution found'

  for *try* := 1 **to** *maxTries* **do**

    a := randomly chosen assignment of the variables in formula *F*;

    for *step* := 1 **to** *maxSteps* **do**

      if *a* satisfies *F* **then return** *a* **end**

      x := randomly selected variable flipping which minimizes the number of unsatisfied clauses;

      a := *a* with *x* flipped;

    end

  end

  return 'no solution found'

end *GSAT*

Basic GSAT (1)

- Simple *iterative best improvement* procedure: in each step, a variable is flipped such that a maximal decrease in the number of unsatisfied clauses is achieved, breaking ties uniformly at random)

  - Uses *static restart mechanism* to escape from local minima

- Terminates when a model has been found, or maxTries sequences of maxSteps variable flips have been performed without finding a model

Basic GSAT (2)

- For any fixed number of restarts, GSAT is *essentially incomplete;* severe stagnation behaviour is observed on most SAT instances
- Provided the basis for many more powerful SLS algorithms for SAT

The GWSAT Algorithm

procedure *GWSAT(F, maxTries, maxSteps)*
input: CNF formula *F*, positive integers *maxTries* and *maxSteps*
output: model of *F* **or** ‘no solution found’
for *try* := 1 **to** *maxTries* **do**
a := randomly chosen assignment of the variables in formula *F*;
for *step* := 1 **to** *maxSteps* **do**
if *a* satisfies *F* **then return** *a* **end**
with probability *1-wp***:** select a variable whose flip minimizes the number of unsatisfied clauses
otherwise: choose a variable appearing in an unsatisfied clause.uniformly at random
a := *a* with *x* flipped;
end
end
return ‘no solution found’
end *GWSAT*

GSAT with Random Walk (GWSAT)

- *Randomised best-improvement procedure* – incorporates *conflict-directed random walk steps* with probability *wp*

- Allows arbitrarily long sequences of random walk steps; this implies that from arbitrary assignment, a model can be reached with a positive, bounded probability, *i.e.*, GWSAT is PAC

- Uses the same static restart mechanism as Basic GSAT

GSAT with Random Walk (continued)

- Substantially outperforms Basic GSAT
- Does not suffer from stagnation behaviour with sufficiently high noise setting; shows exponential RTDs
- For low noise settings, stagnation behaviour is frequently observed

The WalkSAT Architecture

- Based on *2-stage variable selection process* focused on the variables occurring in currently unsatisfied clauses:
  - 1st stage: A clause *c* that is unsatisfied under the current assignment is selected uniformly at random.
  - 2nd stage: one of the variables appearing in *c* is flipped to obtain the new assignment.
- Dynamically determined subset of the GSAT neighbourhood relation – substantially reduced effective neighbourhood size
- Random initialisation and static random restart mechanism as in GSAT

WalkSAT Algorithm Outline

procedure *WalkSAT(F, maxTries, maxSteps, slc)*
    input: CNF formula *F*, positive integers *maxTries* and *maxSteps,*
      heuristic function *slc*
    output: model of *F* **or** ‘no solution found’
    for *try* := 1 **to** *maxTries* **do**
        a := randomly chosen assignment of the variables in formula *F*;
        for *step* := 1 **to** *maxSteps* **do**
            if *a* satisfies *F* **then return** *a* **end**
            c := randomly selected clause unsatisfied under *a;*
            x := variable selected from *c* according to heuristic function *slc;*
            a := *a* with *x* flipped;
        end
    end
    return ‘no solution found’
end *WalkSAT*

Novelty

- Uses a *history-based variable selection mechanism*; based on *age*, *i.e.*, the number of local search steps that have been performed since a variable was last flipped.
- Uses the same scoring function as GSAT.
- Variable selection scheme:
  - If the variable with the highest score does not have minimal age among the variables within the same clause, it is always selected.
  - Otherwise, it is only selected with probability of *1-p*, where *p* is a parameter called *noise setting*.
  - In the remaining cases, the variable with the second-highest score is selected.

Novelty (2)

- Novelty always chooses between the best and second best variable in the selected clause
- Compared to WalkSAT/SKC, Novelty is greedier and more deterministic
- Novelty often performs substantially better than WalkSAT/SKC, but it is *essentially incomplete* and sometimes shows extreme stagnation behaviour.

Novelty$^+$

- By extending Novelty with *conflict-directed random walk* analogously to GWSAT, the essential incompleteness as well as the empirically observed stagnation behaviour can be overcome.

- With probability *1-wp*, Novelty$^+$ selects the variable to be flipped according to the standard Novelty mechanism; otherwise, it performs a random walk step.

- Novelty$^+$ is provably PAC for $wp>0$ and shows exponential RTDs for sufficiently high setting of the primary noise parameter *p*.

WalkSAT with Adaptive Noise

- The performance of WalkSAT algorithms such as Novelty$^+$ critically depends on noise parameter setting
- Optimal noise setting depend on the given problem instance and are typically rather difficult to determine
- *Adaptive WalkSAT* use high noise values only when they are needed to escape from stagnation situations.

Dynamic Local Search Algorithms for SAT

- Most DLS algorithms for SAT are based on variants of GSAT as their underlying local search procedure.

- The penalty associated with clause *c, clp(c),* is updated in each iteration.

- Evaluation function: $g'(F,a) := g(F,a) + \sum_{c \in CU(F,a)} clp(c)$

- Or equivalently: $clw(c) := clp(c) + 1$

$$g'(F,a) := \sum_{c \in CU(F,a)} clw(c)$$

GSAT with Clause Weights

- Weights associated with clauses are initially set to one; before each restart, the weights of all currently unsatisfied clauses are increased by one.

- Underlying local search procedure: a variant of basic GSAT that uses the modified evaluation function.

- Begins each local search phase from a randomly selected variable assignment (different from other DLS methods).

- Performs substantially better than basic GSAT on some instances; with GWSAT as underlying local search procedure, further performance improvements can be achieved.

Exponentiated Subgradient Algorithm (ESG)

- Based on a simple variant of GSAT that in each step selects a variable appearing in a currently unsatisfied clauses whose flip leads to a maximal reduction in the total weight of unsatisfied clauses

- *Scaling stage*: weights of all clauses are multiplied by a factor depending on their satisfaction status.

- *Smoothing stage*: all clause weights are smoothed using the formula $clw(c) := clw(c) \cdot \rho + (1 - \rho) \cdot \overline{w}$

- **Note:** Weight update steps are computationally much more expensive than the weighted search steps.

Scaling and Probabilistic Smoothing (SAPS)

- *Scaling stage* is restricted to the weights of currently unsatisfied clauses; *smoothing* is only performed with a certain probability.

- By applying the expensive smoothing operation only occasionally, the time complexity of the weight update procedure can be substantially reduced.

- Compared to ESG, SAPS typically requires a similar number of variable flips for finding a model of a given formula, but in terms of time performance it is significantly superior to ESG, DLM, and best known WalkSAT variants(except for Novelty^{+}, which performs better in some cases).