Propositional Satisfiability and Constraint Satisfaction

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

Outline:

1. The Satisfiability Problem (SAT)
2. The GSAT Architecture
3. The WalkSAT Architecture
4. Dynamic Local Search Algorithms for SAT
5. Constraint Satisfaction Problems (CSP)
6. SLS Algorithms for CSPs

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)

Alternative Formulations of SAT (1)

- Represent truth values by integers (*true*=1; *false*=0)

- $I(x) := x$; $I(\neg x) := 1 - x$.

- For $c_i = l_1 \vee l_2 \vee \ldots \vee l_{k(i)}$

  $I(c_i) = I(l_1) + I(l_2) + \ldots + I(l_{k(i)})$.

- For $F = c_1 \wedge c_2 \wedge \ldots \wedge c_m$

  $I(F) = I(c_1) * I(c_2) * \ldots * I(c_m)$.

- A truth assignment satisfies c_i iff $I(c_i) \geq 1$

Alternative Formulations of SAT (2)

- $u_i\,(F,a)$:= 1 if clause c_i of F is unsatisfied under assignment a, and $u_i\,(F,a)$:= 0 otherwise.
- $U\,(F,a) := \Sigma_{\mathrm{i=1..m}} u_i\,(F,a)$.
- A model of F corresponds to a solution of

$$a^* \in \arg\min_{a \in \{0,1\}^n} U(F,a)$$

Subject to: $\forall i \in \{1,2,\ldots,m\} : u_i(F,a) = 0$

(Special case of the *0-1 Integer Programming Problem*)

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.

Unary and Binary Failed Literal Reduction

- *Unary failed literal reduction:* If setting a variable x occurring in the given formula F to true makes F unsatisfiable (*i.e.*, adding a unit clause $c{:=}x$ to F and simplifying results in an empty clause) then adding a unit clause $c := \neg x$ to F yields a logically equivalent formula F'.

- *Binary failed literal reduction* works similarly, but for pairs of literals.

Randomly Generated SAT Instance

- Random clause length model (also called *fixed density model*):
  - n variables and m clauses: for each clause, each of the $2n$ literals are chosen with fixed probability p.
- Fixed clause length model (also known as *Uniform Random k-SAT*):
  - n variables, m clauses, and clause length k: for each clause, k literals are chosen uniformly at random from $2n$ literals.

Random k-SAT Hardness and Solubility Phase Transition

For Uniform Random 3-SAT with a given number of variables n, the probability of generating a satisfiable formula depends on the number of clauses, m:

- when m is small, formulae are underconstrained and tend to be *satisfiable*;
- when m is large, formulae are overconstrained and tend to be *unsatisfiable*.

- At some critical value of *m*, formulae tend to be satisfiable with 50% probability (for Uniform Random-3-SAT: k=4.25)

- It has been shown empirically that problem instances from this *phase transition region* of Uniform Random k-SAT tend to be hard to solve.

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

GSAT with Tabu Search (GSAT/Tabu)

- Based on *simple tabu search*: After a variable *x* has been flipped, it cannot be flipped back within the next *tt* steps
- For sufficient high *tt* settings, GSAT/Tabu does not suffer from stagnation behaviour, and for hard problem instances, it shows exponential RTDS.
- It is not clear whether GSAT/Tabu with fixed cutoff parameter *maxSteps* has the PAC property.
- When using instance-specific optimised tabu tenure settings, GSAT/Tabu typically performs significantly better than GWSAT.

HSAT and HWSAT

- When in a search step there are several variables with identical score (*i.e.,* reduction in number of unsat clauses), HSAT always selects the *least recently flipped* variable.

- Although HSAT typically outperforms basic GSAT, it is more likely to get stuck in local minima.

- HWSAT = HSAT extended with random walk mechanism.

- HWSAT is PAC (like GWSAT)

The WalkSAT Architecture

- Based on *2-stage variable selection process* focused on the variables occurring in currently unsatisfied clauses:
  - 1^{st} stage: A clause *c* that is unsatisfied under the current assignment is selected uniformly at random.
  - 2^{nd} 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*

WalkSAT/SKC

- Uses scoring function $score_b(x)$:= number of currently satisfied clauses that become unsatisfied when flipping variable x.

- Variable selection scheme:
  - if there is a variable with $score_b(x)=0$ in the clause selected in stage 1, this variable is flipped (*zero damage step*)
  - if no such variable exists, with a certain probability $1\text{-}p$, a variable with minimal score value is selected uniformly at random (*greedy step*)
  - else (*i.e.,* with probability p = *noise setting*), one of the variables from c is selected uniformly at random.

WalkSAT/SKC (2)

- PAC when applied to 2-SAT; unknown in general case.
- In practice, WalkSAT/SKC with sufficiently high noise setting does not appear to suffer from any stagnation behaviour, and its runtime behaviour is characterized by exponential RTDs.
- Stagnation behaviour is observed for low noise settings.
- With optimised noise setting, WalkSAT/SKC probabilistically dominates GWSAT in terms of the number of variable flips, but not HWSATor GSAT/Tabu; in terms of CPU times, it typically outperforms all GSAT variants.

WalkSAT with Tabu Search (WalkSAT/Tabu)

- Similar to WalkSAT/SKC; additionally enforces a tabu tenure of *tt* steps for each flipped variable.

- If the selected clause *c* does not allow a zero damage step, WalkSAT/Tabu picks the one with the highest score of all the variables occurring in c that are not tabu.

- When all variables appearing in *c* are tabu, no variable is flipped (*null-flip*)

WalkSAT/Tabu (2)

- WalkSAT/Tabu with fixed *maxTries* parameter has been shown to be *essentially incomplete*.

- With sufficient high tabu tenure settings, WalkSAT/Tabu's run-time behaviour is characterised by exponential RTDs; but there are cases in which extreme stagnation behaviour is observed.

- Typically, WalkSAT/Tabu performs significantly better than WalkSAT/SKC.

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.

Methods using Rapid Weight Adjustments

- Benefit from discovering which clauses are most difficult to satisfy relative to recent assignments.
- *WGSAT*: uses the same weight initialisation and update procedure as GSAT with Clause Weights, but updates clause weights after each GSAT step.
- *WGSAT with Decay*: uniformly decreases all clause weights in each weight update phase before the weights of the currently unsatisfied clauses are increased.
- In terms of CPU time, WGSAT typically does not reach the performance of GWSAT or WalkSAT algorithms.

Guided Local Search for SAT (GLSSAT)

- Flip the least recently flipped variable that leads to a *strict decrease* in the total penalty of unsatisfied clauses (if no such variable exists, consider non-increasing flips).

- Performs a complete pass of unit propagation before search begins.

- The penalties of all clauses with maximal utilities are incremented by one after each local search phase, *i.e.*, when a local minimum is encountered.

- *GLSSAT2*: all clause penalties are multiplied by a factor of 0.8 after every 200 penalty updates.

Discrete Lagrangian Method (DLM)

- Underlying local search procedure is based on GSAT/Tabu with clause weights.
- Local search phase ends when the number of neighbouring assignments with larger or equal evaluation function value exceeds a give threshold.
- Clause penalties are initialised to zero, increased by one after each local search phase, and decreased by one occasionally.
- Variants use additional mechanisms for preventing search stagnation based on long term memory.

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).

Constraint Satisfaction Problems (CSP)

- A *CSP instance* is defined by:
  - a set of variables,
  - a set of possible values (or *domain*) for each variable,
  - a set of constraining conditions (*constraints*) involving one or more of the variables.

- The *Constraint Satisfaction Problem (CSP)* is to decide for a given CSP instance whether all variables can be assigned values from their respective domains such that all constraints are simultaneously satisfied.

- CSP instances with at least one solution exists are called *consistent*, while instances that do not have any solutions are called *inconsistent*.

- In a *finite discrete CSP instance* all variables have discrete and finite domains.

- Finite discrete CSP can be seen as generalisation of SAT and is therefore NP-complete.

- Many combinatorial problems can be modelled quite naturally as CSPs.

Encoding CSP instances into SAT

- Sparse encoding (*unary transform* or *direct encoding*):

  $c_{i,v}$ *represents* $x_i := v$.

(1) $\neg c_{i,v1} \vee \neg c_{i,v2} \quad (1 \leq i \leq n; v1, v2 \in D(x_i); v1 \square v2)$

(2) $c_{i,v0} \vee c_{i,v1} \vee \ldots \vee c_{i,vk-1} \quad (1 \leq i \leq n)$

(3) $\neg c_{i1,v0} \vee \neg c_{i,v1} \vee \ldots \vee \neg c_{i,vk-1} \quad \textit{where } (x_{i1} := v1; x_{i2} := v2; \ldots; x_{is} := vs)$

$\textit{violates some constraint } C_j \in C \textit{ with } \sigma(C_j) := s.$

- Other, more compact encodings exist (*e.g.*, *log encoding*, *multivalued encoding*).

CSP Simplification and Local Consistency Techniques

- Local consistency techniques can reduce the effective domains of CSP variables by eliminating values that cannot occur in any solution.

- *Example:* Arc consistency (eliminates values that do not occur in partial assignments satisfying individual constraints)

- Can be used as preprocessing for other CSP algorithms (such as SLS algorithms).

Prominent Benchmark Instances for the CSP

- Uniform Random Binary CSP
- Graph Colouring Problem
- Quasigroup Completion Problem
- n-Queens Problem

SLS Algorithms for Solving CSPs

- **Three types of algorithms:**
  - SLS algorithms for SAT applied to SAT-encoded CSP instances
  - Generalisations of SLS algorithms for SAT
  - Native SLS algorithms for CSPs

“Encode and Solve as SAT” approach

- *Advantage*: Can use highly optimised and efficiently implemented “of-the-shelf” SAT solvers.

- *Potential disadvantage*: May not be able to fully exploit the structure present in given CSP instances.

Pseudo-Boolean CSP and WSAT(PB)

- *Pseudo-Boolean CSP*, or (Linear) Pseudo-Boolean Programming, is a special case of discrete finite CSP, where all variables have Boolean values.

- *WSAT(PB)* is based on a direct generalisation of WalkSAT architecture to Pseudo-Boolean CSP.

- The evaluation function is based on the notion of the *net integer distance* of a constraint from being satisfied.

Pseudo-Boolean CSP and WSAT(PB)

- In each search step:

  - select a violated constraint *C* uniformly at random

  - flip a variable in *C* that leads to *largest decrease in evaluation function value*;
  if no such variable exists, choose the *least recently flipped variable* with probability *wp*;
  otherwise, flip variable such that increase in the evaluation function is minimal.

- Additionally, use simple tabu mechanism.

WalkSAT Algorithms for Many-Valued SAT

- *Non-Boolean SAT*: non-Boolean literal is of the form *z/v* or *~z/v*, where *z* is a variable and *v* a value from the domain of *z*.

- Constraints are in the form of CNF clauses on non-Boolean literals.

- SLS algorithms for SAT, such as WalkSAT, can be generalised to NB-SAT (and slightly more general variants of many-valued SAT) in a straightforward way.

Min Conflicts Heuristic (MCH) and Variants

- MCH iteratively modifies the assignment of a single variable in order to minimise the number of violated constraints.

  - *Iinitialisation*: uniform random picking

  - *Variable selection*: uniformly at random from the *conflict set*, *i.e.*, the set of all variables that appear in a currently unsatisfied constraint.

  - *Value selection*: the number of unsatisfied constraints (conflicts) is minimised.

- MCH is essentially incomplete.

WMCH and TMCH

- WMCH is a variant of MCH that uses a random walk mechanism analogous to GWSAT.
- WMCH is PAC for noise setting > 0.
- TMCH = MCH extended with a simple tabu search mechanism that associates tabu status with pairs (x,v) of variables and values.

A Tabu Search Algorithm for CSP

- TS-GH by Galinier and Hao:
  - Amongst all pairs *(x,v')* such that variable *x* appears in a currently violated constraint and *v'* is any value from the domain of *x*, TS-GH chooses the one that leads to a *maximal decrease* in the number of violated constraints.
  - Augmented with the same tabu mechanism used in TMCH.

- The performance of TS-GH crucially relies on the use of an incremental updating technique (analogous to the one used by GSAT).

- Empirical studies suggest that when applied to the conventional CSP, TS-GH generally achieves better performance than any of the MCH variants.