Satisfiability

Satisfiability, the first NP-complete problem, is a classic problem in constraint satisfaction. In this lecture, we describe complete and incomplete algorithms designed to solve satisfiability. Given a formula of propositional logic, complete methods, such as Davis–Putnam, are guaranteed to find a satisfying assignment, if one exists; on the other hand, incomplete methods such as GSAT and WalkSAT, which are based on local search techniques, need not return a satisfying assignment even if one exists, but perform well in practice.

Let $\mathcal{A} = \{x_1, \ldots, x_n\}$ be an alphabet of propositional variables. Elements of this alphabet are called *positive* literals. For each positive literal $x \in \mathcal{A}$, there exists the corresponding *negative* literal $\neg x$. A set of literals, both positive and negative, forms a *clause*, and a *formula* consists of a set of clauses. By convention, a clause is interpreted as the disjunction of its literals, whereas a formula is interpreted as the conjunction of its clauses. Formulas expressed in this way are said to be in *conjunctive normal form*.

A *truth assignment* is a function $v : \mathcal{A} \to \{t, f\}$. A truth assignment over propositional variables can be extended to a truth assignment over literals as follows: if $v(x) = t$, then $v(\neg x) = f$; otherwise, if $v(x) = f$, then $v(\neg x) = t$. Given a truth assignment, a literal is said to be satisfied iff it is assigned the value t. A conjunction is satisfied iff all its conjuncts are satisfied. A disjunction is satisfied iff at least one of its disjuncts are satisfied. Given a CNF formula ϕ of propositional logic, the satisfiability problem (SAT) is:

$$\boxed{\text{“does there exist a truth assignment under which } \phi \text{ is satisfied?”}}$$

If such an assignment exists, then ϕ is *satisfiable*. Otherwise, ϕ is *unsatisfiable*.

Remark An arbitrary formula of propositional logic constructed using the connectives $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow$ and an alphabet of propositional variables can be systematically converted into a CNF formula.

Example The negation of the formula

$$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$$

can be converted into CNF as follows:

$$\begin{array}{ll} & \neg((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))) \\ \text{iff} & \neg(\neg(\neg A \vee (\neg B \vee C)) \vee (\neg(\neg A \vee B) \vee (\neg A \vee C))) \\ \text{iff} & (\neg A \vee (\neg B \vee C)) \wedge ((\neg A \vee B) \wedge \neg(\neg A \vee C)) \\ \text{iff} & (\neg A \vee \neg B \vee C) \wedge (\neg A \vee B) \wedge A \wedge \neg C \end{array}$$

Some of the most powerful algorithms for solving satisfiability are hill-climbing-style algorithms that view satisfiability as an optimization problem. These algorithms return satisfying assignments when they are found; but they are not guaranteed to find a satisfying assignment, even if one exists.

Consider an instance of SAT with n distinct propositional variables and m disjunctive clauses. Viewed as an optimization problem, the set of states V is the set of truth assignments. If SAT is viewed as a maximization problem, the objective function $f(v) \leq m$ denotes the number of clauses that are satisfied in state v; alternatively, if SAT is viewed as a minimization problem, $f(v) \geq 0$ denotes the number of clauses that are *not* satisfied in state v. We take the point of view of minimization in this lecture.

GSAT is one specialization of hill-climbing that is tailored to the satisfiability problem. States are assignments of the n propositional variables to $\{0, 1\}$. A convenient representation for such an assignment is a bit string $b_n \in \{0,1\}^n$, where b_i denotes the truth value of the ith propositional variable. Based on this state representation, GSAT uses the following neighborhood operation. $\mathcal{N}(v)$ is the set of states that are reachable from v via exactly one bit flip: i.e.,

$$\mathcal{N}(v) = \{u \in V \mid \text{HammingDistance}(u, v) = 1\}$$

Given start state $v = \{v(P) = t, v(Q) = f, v(R) = t, v(S) = f, v(T) = t\}$, we now trace the behavior of GSAT on the following formula:

$$(P \vee Q \vee R) \wedge (\neg P \vee R \vee \neg T) \wedge (Q \vee \neg R \vee S) \wedge (\neg R \vee S \vee \neg T) \wedge (P \vee R \vee T)$$

Representing the start state as a bit string yields $v = 10101$. At state v, (assuming minimization) the objective function $f(v) = 2$, and

$$\mathcal{N}(v) = \{00101, 11101, 10001, 10111, 10100\}$$

The truth values of the clauses are listed below, for each successor state. All clauses are satisfied at state 10111: i.e., $f(10111) = 0$. Therefore, GSAT terminates after a single iteration at optimal state 10111.

$u \in \mathcal{N}(v)$	$P \vee Q \vee R$	$\neg P \vee R \vee \neg T$	$Q \vee \neg R \vee S$	$\neg R \vee S \vee \neg T$	$P \vee R \vee T$	f
00101	T	T	F	F	T	2
11101	T	T	T	F	T	1
10001	T	F	T	T	T	1
10111	T	T	T	T	T	0
10100	T	T	F	T	T	1

In practice, GSAT implements the "force-best-move" heuristic, which forces it to accept some move, even if the best-move beyond the current state is not in fact an improvement. This approach enables the algorithm to proceed beyond local optima. The GSAT algorithm is depicted in Table 1.

GSAT(ϕ, N, M)

Inputs	CNF formula ϕ
	number of restarts N
	number of trials per restart M
Output	satisfying assignment v or `fail`

for $i = 1$ to N

1. initialize random start state: i.e., random assignment v
   (a) for $j = 1$ to M
      i. if v satisfies ϕ, `return` v
      ii. compute the neighborhood of v, $\mathcal{N}(v)$
      iii. let $v \in \arg\min_{u \in \mathcal{N}(v)} f(u)$

`fail`

Table 1: GSAT: Selman, Levesque, and Mitchell [1992]. If an incomplete search method like GSAT fails, then no satisfying assignment was found—however, a satisfying assignment might still exist.

GSAT with random walks is inspired by Papadimitrou's random walk algorithm for 2SAT,[1] which finds a satisfying assignment in $O(n^2)$ bit flips for any 2SAT formula with n variables, with probability 1. Given 2SAT formula ϕ,

[1] The kSAT problem restricts the number of literals per clause to k. 2SAT can be solved in polynomial time, but 3SAT is NP-complete.

REPEAT

- choose unsatisfied clause $C \in \phi$ at random
- choose variable $x \in C$ at random
- flip the assignment of x

UNTIL all clauses are satisfied

GSAT + WALK, that is, with random walks, acts like GSAT with probability p, but otherwise chooses an unsatisfied clause $C \in \phi$ at random; chooses a variable $x \in C$ at random; and flips the assignment of x. (See Table 2.)

Empirically, one of the best algorithms (as of 2005) for solving SAT is known as WALKSAT. Like GSAT + WALK, WALKSAT flips the assignment of some variable x that appears in an unsatisfied clause C. Doing so instantly renders C satisfied, but also has a tendency to render some previously satisfied clauses unsatisfied. The *break-value* of a variable x at state v is defined as the number of clauses that are satisfied by v but break when the value of x is flipped. With probability p, WALKSAT greedily flips the value of a variable $x \in C$ of minimal break value, for some unsatisfied clause $C \in \phi$. Otherwise, WALKSAT flips the value of a random variable $x \in C$.

GSAT+WALK(ϕ, N, M, p)	
Inputs	CNF formula ϕ
	number of restarts N
	number of trials per restart M
	probability p
Output	satisfying assignment v or `fail`

for $i = 1$ to N

1. initialize random start state: i.e., random assignment v
   (a) for $j = 1$ to M
      i. if v satisfies ϕ, `return` v
      ii. with probability p
         A. compute the neighborhood of v, $\mathcal{N}(v)$
         B. let $v \in \arg\min_{u \in \mathcal{N}(v)} f(u)$
      iii. with probability $1 - p$
         A. choose unsatisfied clause $C \in \phi$ at random
         B. choose variable $x \in C$ at random
         C. let $v \leftarrow v$ with bit x flipped

`fail`

Table 2: GSAT+WALK [Selman, Kautz, and Cohen, 1994].

WALKSAT(ϕ, N, M, p)	
Inputs	CNF formula ϕ
	number of restarts N
	number of trials per restart M
	probability p
Output	satisfying assignment v or `fail`

for $i = 1$ to N

1. initialize random start state: i.e., random assignment v
   (a) for $j = 1$ to M
      i. if v satisfies ϕ, `return` v
      ii. choose unsatisfied clause $C \in \phi$ at random
      iii. with probability p
         A. choose a variable $x \in C$ of minimal break-value
      iv. with probability $1 - p$
         A. choose a variable $x \in C$ at random
      v. let $v \leftarrow v$ with bit x flipped

`fail`

Table 3: WALKSAT [Selman, Kautz, and Cohen, 1994].