

Constraint Satisfaction¹

The topic of this lecture is constraint satisfaction problems (CSPs). Examples of CSPs include n -queens, map coloring, and crossword puzzles. We describe two approaches to constraint satisfaction: systematic search—emphasizing backtracking, consistency techniques, constraint propagation, and variable ordering heuristics—and local search, viewing constraint satisfaction as optimization. Satisfiability is the canonical example of a CSP. As such, we present special-purpose algorithms for solving satisfiability: systematic approaches, namely the Davis-Putnam and Davis-Putnam-Logemann-Loveland procedures; and local search methods, including GSAT and WALKSAT.

1 Constraint Satisfaction Problem

A *constraint satisfaction problem* is characterized as follows:

- a (finite) set of variables x_1, \dots, x_n
- a set of (finite) domains D_1, \dots, D_n , one per variable
- a (finite) set of constraints, or relations P_1, \dots, P_m (e.g., $x_1 \neq x_2$; $\neg x_1 \vee x_2$)

A solution to a constraint satisfaction problem is an assignment to each variable an element in its domain *s.t.* all constraints are satisfied.

As stated, a CSP involves “hard” constraints—constraints that either pass or fail, with no middle ground. Alternatively, CSPs can also be formulated as optimization problems based on “soft” constraints. In doing so, the objective is often to maximize the number of constraints satisfied; or equivalently, to minimize the number of constraints violated. Local search algorithms (see Lecture #4) are readily applicable to CSPs formulated in this manner.

Example A classic example of a CSP is the n -queens problem. In this problem, n -queens are to be placed on an $n \times n$ chess board *s.t.* no two queens threaten each other: *i.e.*, no two queens occupy the same row, column, or diagonal.

In the case of 4-queens, the problem can be represented as follows.

- variables x_1, x_2, x_3, x_4 , where variable x_i denotes the row of the queen placed in the i th column

¹Copyright© Amy Greenwald, 2001–04

- domains $D_1 = D_2 = D_3 = D_4 = \{1, 2, 3, 4\}$
- constraints:
 - no two queens can occupy the same row: *i.e.*, $x_1 \neq x_2, x_1 \neq x_3, x_1 \neq x_4, x_2 \neq x_3, x_2 \neq x_4, x_3 \neq x_4$
 - no two queens can occupy the same column: this constraint is automatically satisfied in this formulation of the problem
 - no two queens can occupy the same diagonal: *i.e.*, $|x_1 - x_2| \neq 1, |x_1 - x_3| \neq 2, |x_1 - x_4| \neq 3, |x_2 - x_3| \neq 1, |x_2 - x_4| \neq 2, |x_3 - x_4| \neq 1$

The assignment $\{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 4, x_4 \mapsto 2\}$ solves the 4-queens problem. A solution to the 8-queens problem is depicted in Figure 1.

				Q			
		Q					
Q							
						Q	
	Q						
							Q
					Q		
			Q				

Figure 1: A solution to the 8-queens problem.

2 Systematic Search

In this section, we discuss two forms of systematic search: generate-and-test and backtracking. Generate-and-test is trivial to implement, but impractical. The algorithm systematically generates all complete variable assignments and test all constraints until either a solution is discovered or there are no further variable assignments to test. This algorithm can be implemented as a sequence of nested loops, with each one systematically assigning values to one variable; in the innermost loop, all constraints are tested.

In search with backtracking, variables are assigned values in their domain, one by one, until some constraint can be tested. Upon testing, if the test passes, the variable assignment is extended. Otherwise, if the test fails, then the search backtracks and assigns new values to variables. This form of backtracking is called *chronological backtracking*, since variables are unbound in reverse order. An alternative, *intelligent backtracking*, backtracks to one of the causes of failure, skipping intermediate variables.

3 Satisfiability

Satisfiability, the first problem known to be NP-complete, 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 $\mathcal{A} = \{x_1, \dots, x_n\}$ 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 $v : \mathcal{A} \rightarrow \{t, f\}$. 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) can be stated as follows:

“does there exist a truth assignment under which ϕ is satisfied?”

If such an assignment indeed exists, then ϕ is said to be *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:

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

The logical laws used in this derivation are as follows: $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$ (line 2); the law of double negation: $\neg\neg A \leftrightarrow A$ (line 3); and, DeMorgan’s law: $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$ (lines 3 and 4). \square

3.1 Systematic Search

The original satisfiability algorithm, due to Davis and Putnam, is based on the resolution inference rule of propositional logic (see Lecture #):

$$\frac{x \vee y_1 \vee \dots \vee y_n \quad \neg x \vee z_1 \vee \dots \vee z_n}{y_1 \vee \dots \vee y_n \vee z_1 \vee \dots \vee z_n}$$

An improvement upon this algorithm, DPLL (Davis, Putnam, Logemann, and Loveland), is based on the unit resolution inference rule of propositional logic:

$$\frac{x \vee y_1 \vee \dots \vee y_n \quad \neg x}{y_1 \vee \dots \vee y_n}$$

These two inference rules are sound (but not complete): *i.e.*, if an application of resolution transforms ϕ into ϕ' , then ϕ' is satisfiable whenever ϕ is satisfiable.

3.1.1 Davis–Putnam

Given a formula ϕ , the Davis-Putnam (DP) procedure tackles the satisfiability problem by attempting to apply the resolution rule to each variable in the set $\text{VAR}(\phi)$ in turn (see Table 1). DP simplifies ϕ to ϕ' , until either ϕ' contains the empty clause, which is *not* satisfiable, implying that ϕ itself is *not* satisfiable; or ϕ' can be simplified no further but does not contain the empty clause, implying that ϕ itself is satisfiable. In addition to the application of resolution, after resolving clauses of the form $\{x, y\}$ and $\{\neg x, y\}$ on variable x , DP simplifies the resolvent $\{y, y\}$ to $\{y\}$; and after resolving clauses of the form $\{x, \neg y\}$ and $\{\neg x, y\}$ on variable x , DP omits the resolvent $\{\neg y, y\}$ since it is tautological.

A *tautology* is a formula which is true under all truth assignments. A formula is a tautology iff its negation is unsatisfiable. We now use Davis-Putnam to show that $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$ is a tautology. The clausal form of the negation of this formula is given by: $\{\{\neg A, \neg B, C\}, \{\neg A, B\}, \{A\}, \{\neg C\}\}$. First select variable A . Resolving all clauses that contain A and $\neg A$ yields: $\{\{\neg B, C\}, \{B\}, \{\neg C\}\}$. Next select variable B . Resolving all clauses that contain B and $\neg B$ yields: $\{\{C\}, \{\neg C\}\}$. Lastly, select variable C . Resolving all clauses that contain C and $\neg C$ yields the empty clause. Thus, the negation of this formula is *unsatisfiable*: *i.e.*, the formula is a tautology.

3.1.2 Davis–Putnam–Logemann–Loveland

Unlike the original Davis-Putnam algorithm, which cannot construct a satisfying assignment, DPLL does return a satisfying assignment, if one exists. (In fact, with the appropriate extensions, DPLL is capable of returning *all* satisfying assignments.) DPLL incorporates the following three ideas, which specialize forward checking in CSPs to satisfiability. Note that DPLL cannot introduce redundancies or tautologies, since it is based on unit resolution.

DP(ϕ, v)	
Inputs	CNF formula ϕ truth assignment v
Output	SATISFIABLE or UNSATISFIABLE
Initialize	$N =$ number of propositional variables in ϕ
<ol style="list-style-type: none"> 1. for $i = 1$ to N <ol style="list-style-type: none"> (a) choose variable $x \in \text{VAR}(\phi)$ (b) let the set of resolvents $R = \emptyset$ (c) for all $C_1, C_2 \in \phi$ s.t. $x \in C_1, \neg x \in C_2$ <ol style="list-style-type: none"> i. $\psi = \text{resolution}(C_1, C_2)$: i.e., $\psi = C_1 \setminus \{x\} \cup C_2 \setminus \{\neg x\}$ ii. eliminate any redundancies that appear in ψ iii. if ψ is not tautological, $R = R \cup \{\psi\}$ (d) $C_x = \{C \in \phi \mid x \in C \text{ or } \neg x \in C\}$ (e) $\phi = \phi \setminus C_x$ (f) $\phi = \phi \cup R$ 2. if $\{\} \in \phi$, return UNSATISFIABLE else return SATISFIABLE 	

Table 1: Davis–Putnam.

1. **ELIMINATE SINGLETON CLAUSES:** Given singleton clause $\{l\}$, remove all clauses that contain the literal l , including the singleton clause itself, and remove all occurrences of the literal $\neg l$ in other clauses. Moreover, if $l = x$, then let $v(x) = t$; else if $l = \neg x$, then let $v(x) = f$. (This step, which implements unit resolution, is called *unit propagation*.)
2. **ELIMINATE PURE LITERALS:** A *pure* literal is a literal that appears only in positive form or only in negative form; eliminate all clauses containing any pure literals. Moreover, if $l = x$, then let $v(x) = t$; else if $l = \neg x$, then let $v(x) = f$. (This step is called *purification*. Clauses containing pure literals are easily satisfied, and thus can be eliminated.)
3. **BRANCH RECURSIVELY:** Choose some variable $x \in C$ for some $C \in \phi$. Now ϕ is satisfiable iff $\phi \cup \{\{x\}\}$ is satisfiable or $\phi \cup \{\{\neg x\}\}$ is satisfiable. Checking that $\phi \cup \{\{x\}\}$ is satisfiable amounts to setting $v(x) = t$, while checking that $\phi \cup \{\{\neg x\}\}$ is satisfiable amounts to setting $v(x) = f$. Note that this step in the algorithm is nondeterministic.

Since selecting a free variable in DPLL before branching is nondeterministic, various branching heuristics are employed, including: select the variable that occurs most often; select the variable that induces the most unit propagations. Branching decisions can have tremendous impact on the size of the search space.

DPLL(ϕ, v)	
Inputs	CNF formula ϕ truth assignment v
Output	satisfying assignment v or fail
<ol style="list-style-type: none"> 1. if $\{\}$ $\in \phi$, fail 2. if $\phi = \emptyset$, return v 3. if singleton clause $\{l\} \in \phi$, or, if pure literal l appears in ϕ <ol style="list-style-type: none"> (a) if l is positive, let $v(l) = t$ if l is negative, let $v(l) = f$ (b) return (DPLL (REDUCE (ϕ, l), v)) 4. choose unassigned variable $x \in C$ for some $C \in \phi$ 5. return (DPLL ($\phi \cup \{\{x\}\}$, v) \vee DPLL ($\phi \cup \{\{-x\}\}$, v)) 	

Table 2: DPLL. If DPLL fails, then the formula ϕ is unsatisfiable.

REDUCE(ϕ, l)	
Inputs	CNF formula ϕ , literal l
Output	reduced CNF formula ϕ'
Initialize	$\phi' = \emptyset$
for all $C \in \phi$	
<ol style="list-style-type: none"> 1. if $\{l, \neg l\} \cap C = \emptyset$, then $\phi' = \phi' \cup \{C\}$ 2. else if $\neg l \in C$, then $\phi' = \phi' \cup \{C \setminus \{\neg l\}\}$ 3. else if $l \in C$, then skip 	
return ϕ'	

Table 3: DPLL: REDUCE Subroutine.

As an example, consider the formula $\phi = \{\{x, \neg y, \neg z\}, \{\neg x, y, \neg z\}, \{\neg x, \neg y, z\}\}$, which contains no singleton clauses nor pure literals. To branch on the variable x , DPLL recurses on the formula $\phi \cup \{\{x\}\}$, which contains the singleton clause $\{x\}$. DPLL eliminates $\{x\}$, $\{x, \neg y, \neg z\}$, and any occurrences of the literal $\neg x$. Next, DPLL checks the satisfiability of $\phi' = \{\{y, \neg z\}, \{\neg y, z\}\}$, which contains no singleton clauses or pure literals. To branch on the variable y , DPLL recurses on the formula $\phi' \cup \{\{y\}\}$, which contains the singleton clause $\{y\}$. DPLL eliminates $\{y\}$, $\{y, \neg z\}$, and any occurrences of the literal $\neg y$. Next, DPLL checks the satisfiability of $\phi'' = \{\{z\}\}$. Since ϕ'' is easily satisfiable, DPLL returns the satisfying assignment $v(x) = v(y) = v(z) = t$.

3.2 Local Search

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; thus if they fail, no satisfying assignment was found—still, a satisfying assignment might exist.

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 $\text{Obj}(v) \leq m$ denotes the number of clauses that are satisfied in state v ; alternatively, if SAT is viewed as a minimization problem, $\text{Obj}(v) \geq 0$ denotes the number of clauses that are *not* satisfied in state v .

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 i th propositional variable. Based on this state representation, GSAT uses the following neighborhood operation. $N(v)$ is the set of states that are reachable from v via exactly one bit flip: *i.e.*,

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

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 $\text{Obj}(v) = 2$, and

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

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

$u \in N(v)$	$P \vee Q \vee R$	$\neg P \vee R \vee \neg T$	$Q \vee \neg R \vee S$	$\neg R \vee S \vee \neg T$	$P \vee R \vee T$	Obj
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 4.

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>i.e.</i> , random assignment v	
(a) for $j = 1$ to M	
i. if v satisfies ϕ , return v	
ii. let $v \in \arg \min_{u \in N(v)} \text{Obj}(u)$	
fail	

Table 4: 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,² 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 ,

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 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 5.)

Empirically, the best algorithm 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 $x \in C$.

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

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

<p>for $i = 1$ to N</p> <ol style="list-style-type: none"> 1. initialize random start state: <i>i.e.</i>, random assignment v <ol style="list-style-type: none"> (a) for $j = 1$ to M <ol style="list-style-type: none"> i. if v satisfies ϕ, return v ii. with probability p <ol style="list-style-type: none"> A. let $v \in \text{argmin}_{u \in N(v)} \text{Obj}(u)$ iii. with probability $1 - p$ <ol style="list-style-type: none"> 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 <p>fail</p>
--

Table 5: GSAT+WALK [Selman, Kautz, and Cohen, 1994].

Problems

#1 Solve the following cryptarithmic puzzle.

```

  S E N D
+ M O R E
-----
M O N E Y

```

#2 (a) Formulate the n -queens problem as a CSP.
 (b) Show that n -queens is an instance of SAT.

#3 (a) Formulate the map coloring problem as a CSP.
 (b) Show that map coloring is an instance of SAT.

#4 This puzzle is due to Lewis Carroll. A small street has five houses, each one a different color, and each one housing a man of a different nationality. Each man has a different profession, likes a different drink, and has a different pet.

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 <ol style="list-style-type: none"> 1. initialize random start state: <i>i.e.</i>, random assignment v <ol style="list-style-type: none"> (a) for $j = 1$ to M <ol style="list-style-type: none"> i. if v satisfies ϕ, return v ii. choose unsatisfied clause $C \in \phi$ at random iii. with probability p <ol style="list-style-type: none"> A. choose a variable $x \in C$ of minimal break-value iv. with probability $1 - p$ <ol style="list-style-type: none"> A. choose a variable $x \in C$ at random v. let $v \leftarrow v$ with bit x flipped <p>fail</p>
--

Table 6: WALKSAT [Selman, Kautz, and Cohen, 1994].

The following information is given:

The Englishman lives in the red house.
 The Spaniard has a dog.
 The Japanese man is a painter.
 The Italian drinks tea.
 The Norwegian lives in the first house on the left.
 The owner of the green house drinks coffee.
 The green house is on the right side of the white house.
 The sculptor breeds snails.
 The diplomat lives in the yellow house.
 They drink milk in the middle house.
 The Norwegian lives next door to the blue house.
 The violinist drinks fruit juice.
 The fox is in the house next to the doctor's.
 The horse is in the house next to the diplomat's.

Who has the zebra, and who drinks water?

Formulate this puzzle as a CSP and solve.