(propositional) SAT(isfiability)

Overview

- Boolean formulas
- modeling
- solving
- the Sudoku problem

More details

- http://www.cril.univ-artois.fr/~sais/mastersia.html

Boolean formulas

- $\perp$, $\top$ are formulas
- a propositional variable (atom) A_i is a formula
- if ϕ_1 and ϕ_2 are formulas,
  then $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$, $\phi_1 \leftrightarrow \phi_2$
- literal: a propositional variable A_i (positive literal) or its negation $\neg A_i$ (negative literal)
- $Atom(\phi)$: set of variables A_i of ϕ
- formulas can be represented by trees or DAGs

Complexity

- the problem of deciding the satisfiability of a Boolean formula (SAT) is NP-complete (Cook'71)
- most important problems in logic (validity, deduction, equivalence, . . .) can be reduce to SAT, and they are (co) NP-complete

Disjunctive normal form (CNF)

- ϕ is in CNF (clausal) iff ϕ is a conjunction of disjunctions of literals:

$$\bigwedge_{i \in I} (\bigvee_{j_i \in J_i} A_{j_i})$$

- $\bigvee_{j_i \in J_i} A_{j_i}$ is a clause
- CNF is better handled by algorithm
- example: $\phi = (\neg a \vee b) \wedge c \wedge (a \vee b \vee c) \wedge (\neg a \vee b \vee c \vee)$
- every ϕ can be converted into an equivalent $CNF(\phi)$ (can lead to an exponential explosion of the formula)

Modeling a problem: detecting breakdowns of a vehicle

	breakdowns:		clues
p_1:	no more water	i_1:	temperature light is red
p_2:	no more oil	i_2:	charge is positive
p_3:	belt is broken	i_3:	oil light is red
p_4:	current rectifier is cut	i_4:	revolution counter is positive
p_5:	battery is empty	i_5:	engine is running
p_6:	battery has a short-circuit	i_6:	lights work
p_7:	fuse melt		

	utilities
u_1:	coil is powered
u_2:	secondary circuits are powered
u_3:	there is a short circuit

Modeling a problem: detecting breakdowns of a vehicle

- if the temperature light is red, then,
  there is no more water or no more oil or the belt is broken,
  and secondary circuits are powered
  $i_1 \rightarrow ((p_1 \vee p_2 \vee p_3) \wedge u_2)$
  clauses: $(\neg i_1 \vee p_1 \vee p_2 \vee p_3) \wedge (\neg i_1 \vee u_2)$
- oil light is red iff
  secondary circuits are powered
  and there is no more oil or the engine does not run
  $i_3 \leftrightarrow u_2 \wedge (p_2 \vee \neg i_5)$
  clauses:
  $(\neg i_3 \vee u_2) \wedge (\neg i_3 \vee p_2 \vee \neg i_5) \wedge (\neg u_2 \vee \neg p_2 \vee i_3) \wedge (\neg u_2 \vee i_5 \vee i_3)$
- a battery with short-cut is empty
  $p_6 \rightarrow p_5$
  clause: $\neg p_6 \vee p_5$

Modeling a problem: detecting breakdowns of a vehicle

- if charge is positive and battery is in short-circuit, then the fuse melt
  $i_2 \wedge p_6) \rightarrow p_7$
  clause: $\neg i_2 \vee \neg p_6 \vee p_7$
- ...

Applications of SAT

- Constraint satisfaction
  - scheduling, time-tabling
  - temporal reasoning (Allen 1983)
  - circuit design and verification (VLSI) (Larrabee 1992)
  - ...
- Reasoning (deduction)

$$\Sigma \models \alpha \text{ iff } \Sigma \wedge \neg\alpha \text{ is unsatisfiable}$$

- Other reasoning models in AI
  - diagnostic / abduction
  - belief revision
  - ...
- Learning
- Other applications
  - cryptography
  - model checking
  - ...

Format: DIMACS

- every line starting with a c is a comment
- a line gives the parameters of the model:
  p cnf #variables #clauses
- a variable is given by its number n, it negation by $-n$
- one clause per line, with a 0 at the end
- example: $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge \neg x_3$

```
c dummy program
p cnf 3 4
1 -2 0
2 3 0
1 2 3 0
-3 0
```

Resolution of SAT problems

- numerous solvers
- Davis-Putnam (DP 1960): based on resolution
  $(x_1 \vee A) \wedge (\neg x_1 \vee B) \leftrightarrow (A \vee B)$
- Davis-Putnam-Logeman-Loveland (DPLL 1962): based on backtracking
- modern solvers are based on DPLL with learning
- Some solvers: minisat (sudo apt-get install minisat) , glucose, ...

Classification of solvers

complete	**uncomplete**
can prove unsat	**cannot prove unsat**
resolution	local search
method of tableaux (truth trees)	simulated annealing
DPLL procedure	taboo search
BDDs	genetic algorithms
...	...

Modeling Sudoku

The problem

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- 81 cell board
- cells with values between 1 and 9
- all values of a raw, of a column, or of a bloc (3×3) are different

The model 1/3

- Encoding cells:
  - Direct encoding / Suport encoding: one variable per value
  - Log encoding : $\lceil log_2 n \rceil$ variables to represent n values
- Direct Encoding: each cell corresponds to 9 Boolean variables associated to 9 possible values
  (true if the cell has this value, false otherwise)
  - $x_{i,j,k}$ corresponds to the cell [i,j] taking value k : 729 variables

The model 2/3

- each cell has one and only one value: at least one value per cell, and at most one
  - at least one: $\bigwedge_i \bigwedge_j \bigvee_k x_{i,j,k}$ $\rightarrow$81 clauses
  - at most one:
    $\bigwedge_i \bigwedge_j \bigwedge_{\substack{k,l \\ k \neq l}} (\neg x_{i,j,k} \vee \neg x_{i,k,l})$ $\rightarrow 9 \times 9 \times (9 \times 8)/2 = 2916$ clauses
    (if k and l are integer, can be limited to $k > l$)

The model 3/3

- All the values of a raw are different (alldiff) :
  - $\bigwedge_i \bigwedge_k \bigwedge_{\substack{j,l \\ j \neq l}} (\neg x_{i,j,k} \vee \neg x_{i,l,k}) \quad \rightarrow 9 \times 9 \times (9 \times 8/2) = 2916$ clauses
    (if j and l are integer, can be limited to $j > l$)
- idem for columns and blocs
- cells with given value: unitary clause $x_{i,j,val}$