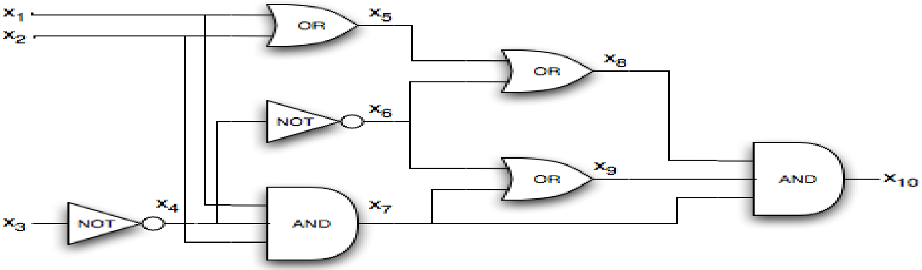
Karlsruhe Institute of Technology

Practical SAT Solving

Lecture 4

Carsten Sinz, Tomáš Balyo | May 9, 2016

INSTITUTE FOR THEORETICAL COMPUTER SCIENCE



KIT – University of the State of Baden-Wuerttemberg and
National Laboratory of the Helmholtz Association

Lecture Outline

- Basic SAT algorithms
  - Stochastic local search
  - Davis-Putnam algorithm
  - DPLL algorithm
  - Stålmarck's method

Repetition: Resolution/Saturation

Saturation Algorithm

- INPUT: CNF formula F
- OUTPUT: $\{SAT, UNSAT\}$

while (true) **do**
  $R = \text{resolveAll}(F)$
  if $(R \cap F \neq R)$ **then** $F = F \cup R$
  else break
if $(\bot \in F)$ **then return** *UNSAT* **else return** *SAT*

Properties of the saturation algorithm:

- it is sound and complete – always terminates and answers correctly
- has exponential time and space complexity

Can we do better?

Karlsruhe Institute of Technology

- Question: Can we do better than saturation-based resolution?
  - Avoid exponential space complexity
  - Improve average-case complexity (for important problem classes)

Stochastic Local Search (SLS)

SAT as an optimization problem: minimize the number of unsatisfied clauses

Start with a complete random assignment α:

0	0	1	0	1	1	0	0	1	1	0	1	0	1	0

Repeatedly flip (randomly/heuristically chosen) variables to decrease the number of unsatisfied clauses:

0	0	1	0	1	0	0	0	1	1	0	1	0	1	0

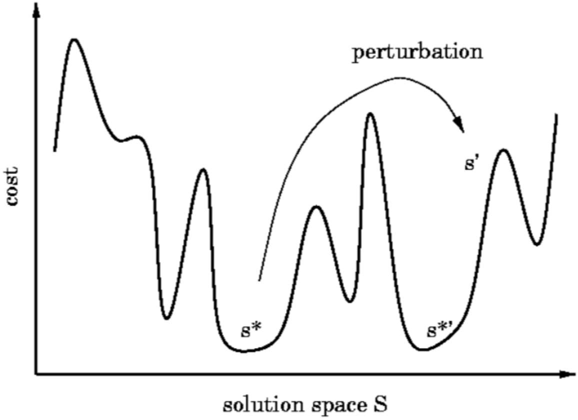
SLS Algorithms

- Local search algorithms are **incomplete**: they cannot show unsatisfiability!
- Many variants of local search algorithms
- Main question: Which variable should be flipped next?
  - select variable from an unsatisfied clause
  - select variable that increases the number of satisfied clauses most
- How to avoid local minima?

GSAT Algorithm [1]

```
Maybe[Assignment] GSAT(ClauseSet S)
{
  for i = 1 to MAX_TRIES do {
    α = random-assignment to variables in S
    for j = 1 to MAX_FLIPS do {
      if ( α satisfies all clauses in S ) return α
      x = variable that produces least number of
        unsatisfied clauses when flipped
      flip x
    }
  }
  return Nothing   // no solution found
}
```

SLS: Illustration



[Source: Alan Mackworth, UBC, Canada]

Walksat [2]

- Variant of GSAT
- Try to avoid local minima by introducing “random noise”
  - Select unsatisfied clause C at random
  - If by flipping a variable $x \in C$ no new unsatisfied clauses emerge, flip x
  - Otherwise:
    - With probability p select a variable $x \in C$ at random
    - With probability $1 - p$ select a variable that changes as few as possible clauses from satisfied to unsatisfied when flipped

SLS: Important Notions

- Consider a flip taking α to α'
- **breakcount:** number of clauses satisfied in α, but not in α'
- **makecount:** number of clauses unsatisfied in α, but satisfied in α'
- **diffscore:** number of unsatisfied clauses in α minus number of clauses unsatisfied in α'
- Typically, **breakcount**, **makecount** and **diffscore** are updated after each flip
- Question: How can we do this efficiently?

GSAT and Walksat Flip Heuristics

- GSAT: select variable with highest **diffscore**
- Walksat:
  - First randomly select unsatisfied clause C
  - If there is a variable with **breakcount** 0 in C, select it
  - otherwise with probability p select a random variable from C, and with probability $1 - p$ a variable with minimal **breakcount** from C

Runtime Comparison Walksat vs. GSAT

formula			DP	GSAT+w	WSAT
id	vars	clauses	time	time	time
2bitadd_12	708	1702	*	0.081	0.013
2bitadd_11	649	1562	*	0.058	0.014
3bitadd_32	8704	32316	*	94.1	1.0
3bitadd_31	8432	31310	*	456.6	0.7
2bitcomp_12	300	730	23096	0.009	0.002
2bitcomp_5	125	310	1.4	0.009	0.001

Table 4: Comparing an efficient complete method (DP) with local search strategies on circuit synthesis problems. (Timings in seconds.)

formula			DP	GSAT+w	WSAT
id	vars	clauses	time	time	time
ssa7552-038	1501	3575	7	129	2.3
ssa7552-158	1363	3034	*	90	2
ssa7552-159	1363	3032	*	14	0.8
ssa7552-160	1391	3126	*	18	1.5

Table 5: Comparing DP with local search strategies on circuit diagnosis problems by Larrabee (1989). (Timings in seconds.)

[Source: Selman, Kautz, Cohen Local Search Strategies for Satisfiability Testing, 1993]

Davis-Putnam Algorithm [3]

- Presented in 1960 as a procedure for first-order (predicate) logic
- Procedure to check satisfiability of a formula F in CNF
- Three (deduction) rules:
  1. **Unit propagation:** if there is a unit clause $C = \{l\}$ in F, simplify all other clauses containing l
  2. **Pure literal elimination:** If a literal l never occurs negated in F, add the clause $\{l\}$ to F
  3. **Case splitting:** Assume that F is put in the form $(A \vee l) \wedge (B \vee \bar{l}) \wedge R$, where A, B, and R are free of l. Replace F by the clausification of $(A \vee B) \wedge R$
- Apply deduction rules (giving priority to rules 1 and 2) until no further rule is applicable

From Davis' and Putnam's Paper

The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore's routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using the present method in 30 minutes.

DPLL Algorithm: Outline

Karlsruhe Institute of Technology

- DPLL: Davis-Putnam-Logemann-Loveland [4]
- Algorithmic improvements over DP algorithm
- Basic idea: case splitting and simplification
- Simplification: unit propagation and pure literal deletion
- Unit propagation: 1-clauses (unit clauses) fix variable values: if $\{x\} \in S$, in order to satisfy S, variable x must be set to 1.
- Pure literal deletion: If variable x occurs only positively (or only negatively) in S, it may be fixed, i.e. set to 1 (or 0).

Pure Literal Deletion: Example

Karlsruhe Institute of Technology

- Let $F_0 = \{\{x, y\}, \{\neg x, y, \neg z\}, \{\neg x, z, u\}, \{x, \neg u\}\}$.
- All clauses containing y may be deleted, as y occurs only positively in F. This yields:

$$F_1 = \{\{\neg x, z, u\}, \{x, \neg u\}\}$$

- Each solution α_1 of F_1 can be extended to a solution α_0 of F_0 by setting $\alpha_0(y) = 1$.
- Moreover, if F_1 does not possess a solution, then so does F_0.
- Repeating yields $F_2 = \{\{x, \neg u\}\}$ and $F_3 = \emptyset$, thus F_0 is satisfiable.

DPLL Algorithm

```
boolean DPLL(ClauseSet S)
{
  while ( S contains a unit clause {L} ) {
    delete from S clauses containing L;   // unit-subsumption
    delete ¬L from all clauses in S;      // unit-resolution
  }
  if ( ⊥ ∈ S ) return false;              // empty clause?
  while ( S contains a pure literal L )
    delete from S all clauses containing L;
  if ( S = ∅ ) return true;               // no clauses?
  choose a literal L occurring in S;      // case-splitting
  if ( DPLL(S ∪ {{L}} ) return true;      // first branch
  else if ( DPLL(S ∪ {{¬L}} ) return true; // second branch
  else return false;
}
```

DPLL: Implementation Issues

- How can we implement unit propagation efficiently?
- Which literal L to use for case splitting?
- How can we efficiently implement the case splitting step?

“Modern” DPLL Algorithm with “Trail”

```
boolean mDPLL(ClauseSet S, PartialAssignment α)
{
  while ( (S,α) contains a unit clause {L} ) {
    add {L = 1} to α
  }
  if ( a literal is assigned both 0 and 1 in α ) return false;
  if ( all literals assigned ) return true;
  choose a literal L not assigned in α occurring in S;
  if ( mDPLL(S, α ∪ {L = 1} ) return true;
  else if ( mDPLL(S, α ∪ {L = 0} ) return true;
  else return false;
}
```

(S, α): clause set S as “seen” under partial assignment α

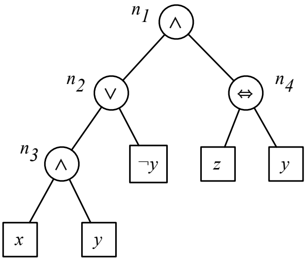
Stålmarck's Method [5]

- Input: Arbitrary formula F in propositional logic (need not be in CNF, $\Rightarrow$ and $\Leftrightarrow$ also allowed)
- Goal: Show unsatisfiability of F
- Preprocessing: Decompose formula tree into simple equations (triplets) T and a literal equivalence class R.
  ($R \subseteq L^0 \times L^0$ where $L^0 = L \cup \{0, 1\}$, R 'consistent')
- Basic processing steps: k-saturation ($k = 0, 1, \ldots$)
  0-saturation: simplification with triplet rules
  k-saturation ($k \geq 1$): case distinction, breadth-first search
- Developed by Gunnar Stålmarck ($\sim$1989), patented

Decomposition into Triplets

$F = ((x \wedge y) \vee \neg y) \wedge (z \Leftrightarrow y)$

Formula tree:



Initial equival. class: $\{n_1 = 0\}$
(to show unsatisfiability of F)

Triplets:

$$n_1 = n_2 \wedge n_4$$
$$n_2 = n_3 \vee \neg y$$
$$n_3 = x \wedge y$$
$$n_4 = z \Leftrightarrow y$$

Normalized triplets:
(only $\wedge$ and $\Leftrightarrow$)

$$n_1 = n_2 \wedge n_4$$
$$\neg n_2 = \neg n_3 \wedge y$$
$$n_3 = x \wedge y$$
$$n_4 = z \Leftrightarrow y$$

Stålmarck's Method: 0-Saturation

Given set of triplets T and literal equivalence class R apply derivation rules (deriving new literal equivalences):

$$\frac{p = q \wedge r \qquad p = 1}{\begin{matrix} r = 1 \\ q = 1 \end{matrix}}\ (A) \qquad \frac{p = q \wedge r \qquad p = \neg q}{\begin{matrix} p = 0 \\ r = 0 \end{matrix}}\ (D)$$

$$\frac{p = q \wedge r \qquad q = 0}{p = 0}\ (B) \qquad \frac{p = q \wedge r \qquad q = r}{p = q}\ (E)$$

$$\frac{p = q \wedge r \qquad q = 1}{p = r}\ (C) \qquad \frac{p = q \wedge r \qquad q = \neg r}{p = 0}\ (F)$$

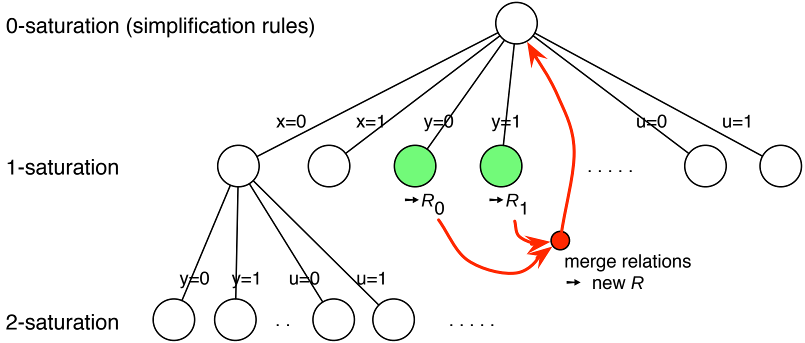
Stålmarck's Method: k-Saturation

Given formula F, represented as (T, R) (triplets and equiv. rel.)
procedure `saturate` extends equivalence relation R:

```
EquivRel saturate(int k, TripletSet T, EquivRel R)
{
  if ( k = 0 ) return zero-saturate(T, R)
  forall x ∈ Var(T) not fixed in R do {
    R0 = saturate(k − 1, T, R ∪ {x = 0})
    R1 = saturate(k − 1, T, R ∪ {x = 1})
    R = R0 ∩ R1
  }
  return R
}
```

(`zero-saturate` returns all-relation if inconsistency was found)

k-Saturation: Graphical Illustration



Summary: Stålmarck's Algorithm

Input: Formula F represented as set of triplets T

(with n_1 representing top of formula tree)

Output: F satisfiable?

```
boolean stalmarck(TripletSet T)
{
  k = 0;  R = {n_1 = 0}
  do {
    R = saturate(k, T, R)
    if ( R = all-relation ) return false
    else if ( R satisfies all triplets T ) return true
    else k = k + 1
  }
}
```

References I

B. Selman, H. Levesque, D. Mitchell, A new method for solving hard satisfiability problems, in: Proceedings of the Tenth National Conference on Artificial Intelligence, AAAI'92, AAAI Press, 1992, pp. 440–446.
URL http://dl.acm.org/citation.cfm?id=1867135.1867203

D. J. Johnson, M. A. Trick (Eds.), Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993, American Mathematical Society, Boston, MA, USA, 1996.

M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (3) (1960) 201–215.
doi:10.1145/321033.321034.
URL http://doi.acm.org/10.1145/321033.321034

References II

M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Commun. ACM 5 (7) (1962) 394–397. doi:10.1145/368273.368557.
URL http://doi.acm.org/10.1145/368273.368557

M. Sheeran, G. Stålmarck, A tutorial on Stålmarck's proof procedure for propositional logic, Formal Methods in System Design 16 (1) (2000) 23–58.
URL http://dx.doi.org/10.1023/A:1008725524946