8. Satisfiability and Model Construction

Davis-Putnam, Phase Transitions, GSAT and GWSAT
Wolfram Burgard & Bernhard Nebel

Motivation (1)


- Brute-force search procedures lead to intelligent behaviour...?
  - ... but these search techniques must be efficient;
  - ... and knowledge is also required (opening- and closing-sequence libraries, good evaluations functions);
  - ... so far, imitating human behavior in chess has not led to any impressive performance.

→ Today’s theme: efficient search techniques for model construction.

Motivation (2)

- Usually:
  - Given: A logical theory (set of propositions)
  - Question: Does a proposition logically follow from this theory?
  - Reduction to unsatisfiability, which is coNP-complete (complementary to NP problems)

- Sometimes:
  - Given: A logical theory
  - Wanted: Model of the theory,
  - Example: Configurations that fulfill the constraints given in the theory.

- Can be “easier” because it is enough to find one model
The Davis-Putnam Procedure

**DP Function**

Given a set of clauses $\Delta$ defined over a set of variables $\Sigma$, return "satisfiable" if $\Delta$ is satisfiable. Otherwise return "unsatisfiable".

1. If $\Delta = \emptyset$ return "satisfiable"
2. If $\square \in \Delta$ return "unsatisfiable"
3. **Unit-propagation Rule:** If $\Delta$ contains a unit-clause $C$, assign a truth-value to the variable in $C$ that satisfies $C$, simplify $\Delta$ to $\Delta'$ and return $\text{DP}(\Delta')$.
4. **Splitting Rule:** Select from $\Sigma$ a variable $\phi$ which has not been assigned a truth-value. Assign one truth value $t$ to it, simplify $\Delta$ to $\Delta''$ and call $\text{DP}(\Delta'')$.
   a. If the call returns "satisfiable", then return "satisfiable"
   b. Otherwise assign the other truth-value to $\phi$ in $\Delta$, simplify to $\Delta'''$ and return $\text{DP}(\Delta''')$.

**Example (1)**

$$\Delta = \{a, b, \neg c, \neg a, \neg b\}, \{c\}, \{a, \neg b\}$$

**Example (2)**

$$\Delta = \{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}$$

**Properties of DP**

- DP is complete, correct, and guaranteed to terminate.
- DP constructs a model, if one exists.
- In general, DP requires exponential time (splitting rule!)
- DP is polynomial on horn clauses, i.e. clauses with at most one positive literal: $($$\neg A_1 \lor \ldots \lor \neg A_n \lor B \equiv \land_i A_i \rightarrow B$$)$
- *Heuristics* are needed to determine which variable should be instantiated next and which value should be used
- In all SAT competitions so far, DP-based procedures have shown the best performance.
Note:
1. The simplifications in DP on Horn clauses always generate Horn clauses
2. A set of Horn clauses without unit clauses is satisfiable
   - All clauses have at least one negative literal
   - Assign false to all variables
3. If the first sequence of applications of the unit propagation rule in DP does not lead to the empty clause, a set of Horn clauses without unit clauses is generated (which is satisfiable according to (2))

4. Although a set of Horn clauses without a unit clause is satisfiable, DP may not immediately recognize it.
   a. If DP assigns false to a variable, this cannot lead to an unsatisfiable set and after a sequence of unit propagations we are in the same situation as in 4
   b. If DP assigns true, then we may get an empty clause - perhaps after unit propagation (and have to backtrack) - or the set is still satisfiable and we are in the same situation as in 4

In summary:
1. DP executes a sequence of unit propagation steps resulting in
   • an empty clause or
   • a set of Horn clauses without a unit clause, which is satisfiable
2. In the latter case, DP proceeds by choosing for one variable:
   • false, which does not change the satisfiability
   • true, which either
     - leads to an immediate contradiction (after unit propagation) and backtracking or
     - does not change satisfiability
3. Run time is polynomial in the number of variables

How Good is DP in the Average Case?
• We know that SAT is NP-complete, i.e. in the worst case, it takes exponential time.
• This is clearly also true for the DP-procedure.
→ Couldn’t we do better in the average case?
• For CNF-formulae in which the probability for a positive appearance, negative appearance and non-appearance in a clause is 1/3, DP needs on average quadratic time (Goldberg 79)!
→ The probability that these formulae are satisfiable is, however, very high.
Phase Transitions ...

Conversely, we can, of course, try to identify hard to solve problem instances

Cheeseman et al. (IJCAI-91) came up with the following plausible conjecture:

All NP-complete problems have at least one order parameter and the hard to solve problems are around a critical value of this order parameter. This critical value (a phase transition) separates one region from another, such as overconstrained and underconstrained regions of the problem space.

Confirmation for graph coloring and Hamilton path ... later also for other NP-complete problems.

Phase Transitions with 3-SAT

Constant clause length model (Mitchell et al., AAAI-92): Clause length \( k \) is given. Choose variables for every clause \( k \) and use the complement with probability 0.5 for each variable.

Phase transition with 3-SAT with a clause/variable ratio of ca. 4.3:

Empirical Difficulty

The Davis-Putnam (DP) Procedure shows extreme running time peaks in this phase transition:

But: Very difficult instances can exist even in the “simple” region of the satisfiable instances!

Notes on the Phase Transition

- When the probability of a solution is close to 1 (under-constrained), there are many solutions, and the first search path of a backtracking search is usually successful.
- If the probability of a solution is close to 0 (over-constrained), this fact can usually be determined early in the search.
- In the phase transition stage, there are many near successes (“close, but no cigar”).
  → (limited) possibility of predicting the difficulty of finding a solution based on the parameters.
  → (search intensive) benchmark problems are located in the phase region (but they have a special structure)
Local Search Methods for Solving Logical Problems

In many cases, we are interested in finding a satisfying assignment of variables (example CSP), and we can sacrifice completeness if we can “solve” much large instances this way.

Standard process for optimization problems: **Local Search**

- Based on a (random) configuration
- Through local modifications, we hope to produce better configurations

→ Main problem: local maxima

Dealing with Local Maxima

As a measure of the value of a configuration in a logical problem, we could use the number of satisfied constraints/ clauses.

But local search seems inappropriate, considering we want to find a global maximum (all constraints/ clauses satisfied).

By **restarting** and/or **injecting noise**, we can often escape local maxima.

**Actually**: Local search performs very well for finding satisfying assignments of CNF formulae (even without injecting noise).

---

**GSAT**

Procedure GSAT

**INPUT**: a set of clauses $\alpha$, MAX-FLIPS, and MAX-TRIES

**OUTPUT**: a satisfying truth assignment of $\alpha$, if found

begin

for $i := 1$ to MAX-TRIES

\[ T := \text{a randomly-generated truth assignment} \]

for $j := 1$ to MAX-FLIPS

if $T$ satisfies $\alpha$ then return $T$

$\nu := \text{a propositional variable such that a change in its truth assignment gives the largest increase in the number of clauses of } \alpha \text{ that are satisfied by } T.$

$T := T \text{ with the truth assignment of } \nu \text{ reversed}$

end for

end for

return “no satisfying assignment found”

end

---

**The Search Behavior of GSAT**

- In contrast to normal local search methods, we must also allow sideways movements!
- Most time is spent searching on **plateaus**.

![Graph showing search behavior]
### Application of GSAT

<table>
<thead>
<tr>
<th>formulas</th>
<th>GSAT</th>
<th>DP</th>
</tr>
</thead>
<tbody>
<tr>
<td>vars</td>
<td>clauses</td>
<td>M-FLIPS</td>
</tr>
<tr>
<td>50</td>
<td>215</td>
<td>250</td>
</tr>
<tr>
<td>100</td>
<td>430</td>
<td>500</td>
</tr>
<tr>
<td>140</td>
<td>602</td>
<td>700</td>
</tr>
<tr>
<td>150</td>
<td>645</td>
<td>1500</td>
</tr>
<tr>
<td>300</td>
<td>1275</td>
<td>6000</td>
</tr>
<tr>
<td>500</td>
<td>2150</td>
<td>10000</td>
</tr>
</tbody>
</table>

### GSAT + Noise

In order to escape from plateaus, we can inject noise.

1. **Simulated annealing (cooling)**
   - With probability $p$, choose a variable in a clause that is still unsatisfied, and change its assignment. With probability $1 - p$, use the GSAT strategy.

2. **Random walk:**
   - Like random walk, but choose any clause (can be already satisfied).

### Results on Random-3CNF Formulae

<table>
<thead>
<tr>
<th>formula</th>
<th>basic</th>
<th>GSAT</th>
<th>Simul. Ann.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vars</td>
<td>clauses</td>
<td>time</td>
<td>flips</td>
</tr>
<tr>
<td>100</td>
<td>430</td>
<td>7554</td>
<td>0.2</td>
</tr>
<tr>
<td>200</td>
<td>860</td>
<td>2849</td>
<td>4</td>
</tr>
<tr>
<td>400</td>
<td>1700</td>
<td>132</td>
<td>$2.6 \times 10^5$</td>
</tr>
<tr>
<td>600</td>
<td>2550</td>
<td>1471</td>
<td>$2.0 \times 10^5$</td>
</tr>
<tr>
<td>800</td>
<td>3400</td>
<td>*</td>
<td>286</td>
</tr>
<tr>
<td>1000</td>
<td>4250</td>
<td>*</td>
<td>1995</td>
</tr>
<tr>
<td>2000</td>
<td>5480</td>
<td>*</td>
<td>3555</td>
</tr>
</tbody>
</table>

- Time in CPU-seconds on an SGI Challenge 100 MHz
- All values are given for the best parameter assignment (restarts, $p$), where walk and noise require no restarts.
- "*" means more than 1000 restarts or more than 20 CPU-hours.

### Results on Circuit Design Problems

Circuit design problems for random circuits to test integer programming approaches.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>vars</td>
<td>clauses</td>
<td>time</td>
<td>flips</td>
<td>time</td>
</tr>
<tr>
<td>11601</td>
<td>1650</td>
<td>2039</td>
<td>114</td>
<td>76985</td>
</tr>
<tr>
<td>11601</td>
<td>1728</td>
<td>78</td>
<td>452</td>
<td>2870019</td>
</tr>
<tr>
<td>11601</td>
<td>1580</td>
<td>768</td>
<td>9.5</td>
<td>12176</td>
</tr>
<tr>
<td>11601</td>
<td>1250</td>
<td>1547</td>
<td>174</td>
<td>572219</td>
</tr>
<tr>
<td>11601</td>
<td>1245</td>
<td>2155</td>
<td>1.7</td>
<td>2090</td>
</tr>
</tbody>
</table>

Similar results for other circuit design and diagnosis problems.
State of the Art

- SAT competitions since beginning of the ‘90
- Current SAT competitions (http://www.satiive.org):
  - 2002:
    - Largest solved instances:
      - 100,000 variables
      - 1,000,000 clauses
    - Smallest unsolved instances:
      - 100 variables
      - 1,000 clause

Concluding Remarks

- Main problem: Determination of the parameters MAX-TRIES, p … but seems connected to problem class.
- Second problem: Cannot find an assignment (that exists) under certain circumstances → incompleteness regarding satisfiability.
  → Area of application: Only for use with very large problem instances for which it is worth determining parameters.
- GWSAT seems to do better than GSAT.
- Tabu-search probably does better than GWSAT – some authors say.
  → At the limit of what can be handled, a bit of “black magic” is required.
  → Local search seems to model our “intuition”.