Motivation

- 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 $\nu$ 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 $\nu$ in $\Delta$, simplify to $\Delta''$ and return $\text{DP}(\Delta'')$. 

Foundations of AI

8. Satisfiability and Model Construction

Davis-Putnam, Phase Transitions, GSAT

Wolfram Burgard, Andreas Karwath, Bernhard Nebel, and Martin Riedmiller
Example (1)
\[ \Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\} \]

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. 
  \((\neg A_1 \lor \ldots \lor \neg A_n \lor B \equiv \land_i A_i \rightarrow B)\)
  \(\rightarrow\) 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.

Example (2)
\[ \Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\} \]

DP on Horn Clauses (1)
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))
DP on Horn Clauses (2)

4. Although a set of Horn clauses without a unit clause is satisfiable, DP may \textbf{not immediately recognize} it.
   a. If DP assigns \textit{false} to a variable, this cannot lead to an unsatisfiable set and after a sequence of unit propagations we are in \textit{the same situation as in 4}.
   b. If DP assigns \textit{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 \textit{the same situation as in 4}.

DP on Horn Clauses (3)

In summary:

1. DP executes a \textbf{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 \textbf{choosing} for one variable:
   - \textit{false}, which does not change the satisfiability
   - \textit{true}, which either
     - leads to an immediate contradiction (after unit propagation) and backtracking or
     - does not change satisfiability

   \textbf{Run time is polynomial} in the number of variables.

How Good is DP in the Average Case?

\begin{itemize}
  \item We know that SAT is NP-complete, i.e., in the worst case, it takes exponential time.
  \item This is clearly also true for the DP-procedure.

→ Couldn’t we do better in the \textbf{average case}?

  \item 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 \textit{quadratic time} (Goldberg 79)!\end{itemize}

→ The probability that these formulae are satisfiable is, however, very high.

Phase Transitions …

Conversely, we can, of course, try to identify \textbf{hard to solve} problem instances.

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

All NP-complete problems have at least \textit{one order parameter and the hard to solve problems are around a critical value of this order parameter. This critical value (a \textit{phase transition}) separates one region from another, such as over-constrained and under-constrained 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 for 3-SAT with a clause/variable ratio of approx. 4.3:

![Graph showing phase transition for 3-SAT](image)

Empirical Difficulty

The Davis-Putnam (DP) Procedure shows extreme runtime peaks at the phase transition:

![Graph showing empirical difficulty](image)

Note: Hard instances can exist even in the regions of the more easily satisfiable/unsatisfiable 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
        $T :=$ a randomly-generated truth assignment
        for $j:=1$ to MAX-FLIPS
            if $T$ satisfies $\alpha$ then return $T$
            $v :=$ a propositional variable such that a change in its truth assignment gives the largest increase in the number of clauses of $\alpha$ that are satisfied by $T$.
            $T:=T$ with the truth assignment of $v$ 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.

State of the Art

- SAT competitions since beginning of the ’90
- Current SAT competitions (http://www.satcompetition.org/):
  In 2007:
  - Largest “industrial” instances: 1,000,000 literals with size 10,000,000
  - Complete solvers are as good as randomized ones!
Concluding Remarks

- DP-based SAT solver prevail:
  - Very efficient implementation techniques
  - Good branching heuristics
  - Clause learning
- Incomplete randomized SAT-solvers
  - are good (in particular on random instances)
  - but there is no dramatic increase in size of what they can solve
  - parameters are difficult to adjust