Scribe Notes: 12/1/09
Presenter: Pingyu Zhang
Scribe: Wesley Botham
Paper: *Propositional Satisfiability and Constraint Programming: A Comparative Survey*
Authors: Lucas Bordeaux, Youssef Hamadi & Lintao Zhang, ACM Computing Surveys, Dec 2006

The speaker

1. Illustrated how to model a logic circuit as a SAT problem
2. Discussed the following basic SAT algorithms: (a) The DP algorithm (Davis, Putnam, 1960); (b) The DPLL algorithm (Davis, Logemann, Loveland, 1962); and (c) A stochastic search algorithm (Selman, Levesque & Mitchell, 1992)
3. Analyzed the components and implementation details of DPLL in terms of (a) branching, (b) propagation and (c) backtracking, and
4. Provided a point-by-point comparison of SAT and CP perspectives on search

For item 3, the speaker related the SAT mechanism to the corresponding one in CP. The speaker also provided a list of new terms that appear in the paper. This summary follows the (above-listed) structure of the presentation but covers only the first half of the talk (i.e., item 3.(b) inclusive). It concludes with a glossary of the terms pertaining to the first half of the presentation.

1. Modeling a Boolean Logic Circuit as a SAT

In SAT, we are asked to determine whether a Boolean sentence can be satisfied. More formally, in conjunctive normal form (CNF), a sentence is a conjunction of clauses where a clause is a disjunction of literals and a literal is a Boolean variable or its negation. The query is: "Is there a simultaneous assignment to all variables such that the sentence evaluates to true?"

The speaker illustrated how to model a hardware verification problem as a SAT problem: determine whether two circuits were logically equivalent under all combinations of inputs. The inputs and outputs of the circuits are modeled as SAT variables, the gates as their equivalent logical sentences. The query was mapped into the requirement that the output of the NXOR gate comparing the outputs of the two circuits be always 1. (Chris noted that an error on the slides: the rightmost gate on slides 6 and 8 is a XOR gate and not an NXOR gate. Pingyu subsequently repaired the slides keeping the XOR gate and replacing the output by 0.)

2. Overview of basic SAT algorithms

Below we summarize the three main procedures for solving SAT.

2.1 DP (Davis-Putnam)

The DP (Davis-Putnam) resolution algorithm consists of a repetitive application of the resolution inference rule in logic. It works as follows:

- Choose a free variable according to some decision heuristic
- Apply the resolution rule with respect to the variable in all clauses where it appears, generating new resolved clauses and adding them to the SAT theory
- Repeat until no more variables can be resolved

Unfortunately, this mechanism may produce an exponential number of resolved clauses, which must be recorded, yielding memory explosion.

2.2 DPLL (Davis-Putnam-Logemann-Loveland)

The Davis-Logemann-Loveland (DLL) algorithm (historically known as DPLL[1]) is basically an exhaustive backtrack search that uses the unit literal rule (see glossary) to propagate variable assignments, the conflict rule (see glossary) to detect unsatisfiability, and an implication graph (see glossary) to enhance backtracking. It works as follows:

- Using some search heuristic (a.k.a. decision heuristic), assign a value to a free variable
- Apply the unit literal rule until no new assignments can be forced. (The instructor reminded the class that it is not necessary to reconsider 'fired clauses' as each clause can be used once.)
- If a conflict is detected, learn a clause that prevents this conflict and backtrack
- Repeat the above steps until a solution is generated or the problem is found to be unsatisfiable

The speaker walked us through the example on slide 15 using a search heuristic that favors increased propagation and early conflict detection.

DPLL avoids the exponential clause generation and memory usage explosion of DP. It is also, in practice, the general frontrunner among SAT search algorithms.

2.3 A stochastic method (e.g., GSAT by Selman, Levesque & Mitchell 1992)

One example of a stochastic method is local search, which we have already seen for CP. It works as follows:

---

[1] Hilary Putnam was not involved with the creation of the DPLL algorithm, but he is included in the algorithm's name for historical reasons.

- Initialize to a random assignment (or 'state') of all variables
- Using a cost function to evaluate the quality of current state and that of neighboring states and a heuristic to move to a neighboring state (e.g., flipping variables in unsatisfied clauses in an attempt to minimize their number).
- When a local minimum is encountered (local search cannot improve the solution), use random walk or other strategies to randomly 'leap' to a neighboring state and repeat,

Unlike previous algorithms, this method is stochastic and therefore not complete.

3. Main steps in DPLL

The DPLL algorithm (sketch shown to the right) has three main components, namely DECIDENEXTBRANCH, DEDUCE, and ANALYZECONFLICTS. The CP analogues of those components are:

```
sat ← false
level ← 0
While sat = false
  If DecideNextBranch
     While Deduce ≠ conflict
        level ← AnalyzeConflicts
        If level < 0
           Return sat
        Else Backtrack(level)
  Else
     Return sat ← true
```

1. DECIDENEXTBRANCH uses a heuristic to decide which variable to instantiate as search proceeds
2. DEDUCE applies a propagation algorithm to perform some kind of look-ahead during search
3. ANALYZECONFLICTS backjumps when a dead end is encountered by analyzing and recording the source of conflict

3.1 Branching: Decision Heuristics

Many heuristics for use by DECIDENEXTBRANCH were discussed, generally falling into the same two strategies used in CP heuristics: pursuing early conflicts and pursuing solutions. For efficiency reasons, sublinear heuristics are preferred. Heuristics discussed included:

- RAND, which chooses a variable completely at random
- Dynamic Largest Combined Sum (DLCS), which chooses the variable which appears in the greatest number of clauses (in CP, this heuristic corresponds to the largest degree heuristic)
- Dynamic Largest Individual Sum (DLIS), which chooses the variable that has the greatest number of literals or the greatest number of complement literals in the problem

In general, upon choosing a variable to instantiate, both DLCS and DLIS will instantiate it to whichever value satisfies the greatest number of unresolved clauses. However, randomized heuristics exist, which simply assign a random value to this variable.

DLIS and DLCS both use expensive methods to maintain statistics on the number of literals satisfied or unsatisfied by a given assignment. Furthermore, they do not account for new learned clauses. The VSIDS (Variable State Independent Decaying Sum) heuristic addresses both shortcomings. VSIDS works as follows:

- When instantiating a new variable, choose the literal that has the highest 'score'
- The score of a literal is the number of times it appears in the problem among all clauses
- When new clauses containing the literal are learned, the score is updated accordingly
- We periodically induce 'decay' by dividing all scores by some constant; that decay gives greater weight in decision-making to recently-learned clauses

We discussed the authors' claim that, unlike SAT decision heuristics, CP search heuristics are domain dependent and highly specific to problems. It was argued that, although tailored heuristics do exist, the bulk of CP search heuristics, in fact, have very general applicability.

3.2 Binary Clause Propagation (BCP)

Binary clause propagation[2] is a powerful method that can be incorporated into the DEDUCE step of DPLL in addition to the unit rule literal rule. It is based on applying the unit resolution rule repeatedly to quickly infer the values of forced variables (reminiscent of the domino effect in a look-ahead scheme in CP).

The speaker stated that BCP takes up to 90% of the time in DPLL and we must therefore be extremely careful in implementing it. For this reason a wide variety of implementations have been developed in hopes of reducing this effort. (The instructor drew a parallel with the myriad arc consistency, path consistency, etc. propagation algorithms that are constantly being fine-tuned by researchers in CP.)

The speaker introduced three leading BCP implementations (GRASP, SATO, and Chaff). The class ended at that point and those BCP algorithms will be the discussed in the following set of minutes.

[2] Do not confuse Binary Clause Propagation (BCP) with Binary Clause Reasoning. See the glossary for further explanation.

Glossary

Binary clause reasoning: 2SAT is tractable; binary clause reasoning applies that insight to identify binary clauses within a SAT problem and to solve this fragment of the theory in polynomial time.

Conflicting rule: If all literals (not variables!) are forced to 0 during propagation, the SAT sentence becomes unsatisfiable. When this situation occurs during search, we know we have reached a dead end: we must identify the source of the conflict and backtrack to it.

Hyper-resolution: The repeated application of the resolution inference rule can treated as a single inference step yielding the same result.

Hyper-bin-resolution: This mechanism is a special instance of hyper-resolution and focuses on generating new binary clauses by the application of the resolution rule in order to allow further application of the binary clause reasoning.

Implication graph: A directed graph in which vertices are variable assignments (either by search or by propagation). The edges record the clauses that caused the assignment of a variable by propagation. The implication graph stores information on the reason for each variable's assignment: whether it was assigned by a search heuristic or by propagation and which level of search it was assigned on.

Learned clause: A clause learned by applying the conflicting rule. It is the negation of the clause which ultimately caused the contradiction to occur.

Original clause: A clause included in the original statement of the problem.

Resolution rule: Consider the case where two (disjunctive) clauses contain the same variable, one in its positive form and the other in its negative form. The resolution rule infers a new (disjunctive) clause listing all the literals in either of the two clauses not including the complementary pair.

Unit literal rule: When, in a disjunctive of i literals, (i-1) literals are set to 0, we can force the single remaining literal to evaluate to 1 (otherwise the SAT sentence is unsatisfiable).

Unit resolution: This rule is a special case of the resolution rule where one clause has only two literals and the other one literal. The application of the unit resolution rule allows us to force the value of the literal appearing in only one of the clauses. The repetitive application of the unit resolution rule is the basis of the Binary Constraint Propagation (BCP) mechanism.

Variables:

- *Free variable*: A variable whose value has not been set.
- *Decision variable*: A variable whose value is set by search.
- *Forced variable*: A variable whose value was determined by propagation.