Scribe Notes of December 10, 2009
Speaker: Pingyu (Part II)
Scribe: Chris

Paper: *Propositional Satisfiability and Constraint Programming: A Comparative Survey*
ACM Computing Surveys, 2006
Authors: LUCAS BORDEAUX, YOUSSEF HAMADI and LINTAO ZHANG

The lecture is a continuation of the previous lecture, which stopped after covering the decision step in DPLL (ref. ordering heuristics in CSP) and before starting Binary Constraint Propagation (BCP, ref. lookahead checking in CP). The lecture started with a review of the material seen in the previous lecture. Then, it proceeded to the discussion of the three major implementations of BCP, and a high-level comparative analysis of SAT and CP (i.e., a synthesis).

1. Review of Concepts and Terms

Concepts

1. DP
→Robert: Named for Davis Putnam, the coolest guy in the world.
2. DPLL: Contains the P for historical reasons. Putnam was not involved in the development of this algorithm.
3. BCP: Binary Constraint Propagation
→Peter: Adds learned clauses
4. DECIDENEXTBRANCH: The step of the search algorithm that does the branching and ordering.
→Professor: Maybe, the SAT equivalent of CP's ordering heuristics.
5. DEDUCE: The pruning and propagation part of the SAT solver algorithm.
→Professor: Maybe, the SAT equivalent of CP's Forward Checking (and domino affect).
6. ANALYZECONFLICT: The backtracking and learning part of the SAT solver algorithm.

Terms

1. Resolution rule:
→Wesley: $(a \vee \neg b) \wedge b \rightarrow a$
→Professor: That is the *unit* resolution.
→Robert: When there are two clauses where a literal is complementary, a third clause can be added with all the literals except the one that is complementary.
2. Unit literal rule:
→Peter: When a term is true.
→Professor: So all terms in a clause are *null*, except one, which is not instantiated. We know that the value of the last term must be true. Otherwise the clause does not hold and neither does the sentence.
3. Conflicting rule:
→Robert: If all literals are 0 in a clause, the problem is unsatisfiable.
4. Decision heuristics:
→Robert: Heuristics to determine variable selection
5. Implication graph:

→Professor: It is a directed graph. First, whenever you define a graph, you must say what the nodes and the edges are. Second, since it is directed, you must also say what the direction on the arrows means. Now, it is a graph where nodes are either decision nodes or enforced nodes. The edges are added after each decision is made to trace which are literals are forced by propagation (e.g., application of the unit literal rule or the unit resolution rule). Whenever a literal is instantiated by propagation, edges are added from the nodes that 'caused' the propagation to the enforced literal. It is also important to keep track of at which level in the search an instantiation occurred, as a label of the node. The graph is used to detect conflicts whenever a literal and its negation are both *enforced*.

6. Original clause:

→Pingyu: A clause specified in the problem.

7. Learned clause:

→Wesley: A clause that is learned from a conflict and added to the set of original clauses to speed up propagation/inference.

8. Binary clause reasoning:

→Professor: What type of SAT is tractable? Class: 2SAT. Professor: Binary clause reasoning identifies the binary clauses in the problem and solves them. Those clauses are parts of the problem that can be solved in polynomial time.

9. Hyper-resolution:

→Pingyu: Process of applying the resolution rule to two non-binary clauses (or one binary and one ternary clause) to infer a new *binary* clause.

2. Three Implementations of the Binary Constraint Propagation (BCP)

The speaker first recalled the BCP mechanism, then discussed three implementations of the BCP as they are done in GRASP, SATO, and Chaff.

2.1 BCP: Pruning by propagation

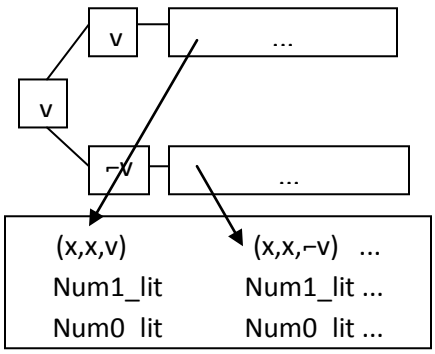
The principal of BCP is the repeated application of the propagation rules (e.g., unit propagation or more involved ones until quiescence or the detection of a conflict. For example:

1. Apply the unit literal rule when all but one of the literals of a clause have been instantiated.
2. Apply the conflict rule when all literals in a clause have been assigned.
3. Ignore the clause as satisfied when one literal in the clause is assigned 1.

We therefore need to monitor the number of positive and negative literals in a clause, and which ones are assigned what, and we need to repeatedly scan the clauses and update those numbers, which can become quite costly.

2.2 Literal Counting in GRASP

Each literal keeps track of the clauses where it appears. And each clause keeps track of the number of literals with the value 1 and the number with value 0 along with a constant that is the total number of literals in the clause. When a variable is assigned, we can easily look up the



clauses where a literal appears and update the counts accordingly. Using the counters we can find when a clause is:

1. Unit: (Num_0_Lits = Num_all_Lits – 1) and (Num_1_Lits = 0)
2. Conflict: Num_0_Lits = Num_all_Lits
3. Satisfied: Num_1_Literals > 0

Improvement: The above implementation can further be improved by removing Num_1_Lit and just keeping a Num_Non_Zero_Literals. Now we can find when a clause is:

1. Unit: Num_Non_Zero_Literals = 1
2. Conflict: Num_Non_Zero_Literals = 0
3. Satisfied:

→Peter: This requires more checking to determine whether the last literal is a one.

→Pingyu: True, but this halves the updating effort.

A variable assignment/unassignment takes *l*m / 2n* operations on average

→ Professor: Big O notation is for the asymptotic upper bound, not for the average-case analysis.

2.3 BCP in SATO

SATO introduces a new data structure, which works with two pointers for each clause. A pointer points to the first/last literal from beginning/end of the clause that is either free (unassigned) or has value 1. Each variable keeps 4 lists: (1) Positive heads: list of clauses where it is a positive head, (2) Negative heads, (3) Positive tails, and (4) Negative tails.

When a variable is assigned, the lists are used to update the pointers.

When a variable is unassigned, all clauses have to be examined.

→The professor pointed out this is similar to restoring the domains when doing a backtrack during a backtrack search. Also, it is similar to updating the 'support' pointers when doing AC2001 or another AC algorithm with residues.

2.4 BCP in Chaff

The idea is to watch only two literals in a clause. Since an N-literal clause only becomes a unit clause when all but one literal has been assigned we can theoretically ignore the first N-2 assignments.

→Rob: Big advantage: During backtracking, no updates need to be made.

→Professor: Round of applause.

Process:

1. Watch the first two literals in any clause
2. Whenever a literal is assigned to 0 examine every clause where it is watched.
    a. If there is an unwatched literal in the clause that is unassigned, swap the two and begin watching the other one.
    b. Otherwise, the other watched literal is now the only unassigned literal and is in a unit clause, add it to the processing queue.

3. Analyzing Conflicts

Mechanism: When a conflict is encountered in the conflict graph, choose a set of variable instantiations that have led to the conflict (a cut), take the contrapositive of the cut, and add it as a new rule.

→Shant: But don't these new rules cease to hold when you backtrack?

→Professor: No, this is not removing legal solutions. The contrapositive always holds.

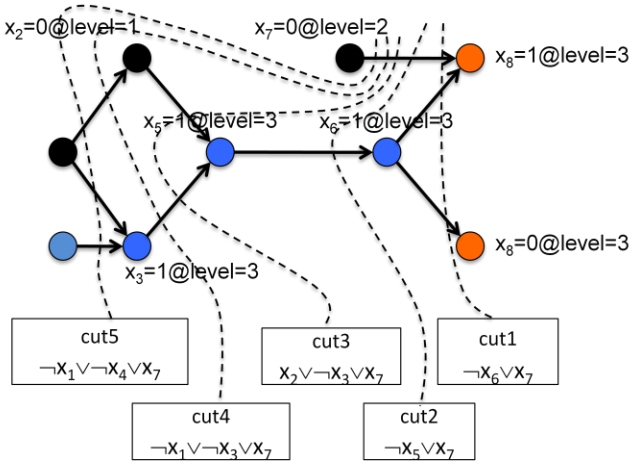
Then backtracking can be done to a level where the conflict/learned clause is no longer false.

→Professor: Does this mechanism remind you of anything?

→Wesley: Conflict directed back jumping.

→Professor: Yes, but this is stronger.

There are several different learned clauses that can be drawn from different cuts for the same conflict.



It is desirable to have a learned clause containing a unique implication point (UIP). A UIP is a node in the decision graph where all paths from the current decision variable to the conflicting variables pass through this node.

3.1 Choosing a learned clause

1. *Decision only scheme*: The learned clause consists of *only* decision variables. This heuristic is flawed since in a systematic search we will never try this set of decisions again.

→ Shant: argued that this would backtrack the furthest

→Professor: It backtracks just until the deepest one of the decision variables is unassigned. Remember that the enforced variables are not 'in the tree.'

2. *RelSat decision scheme*: Take the cut that only contains one variable from the current level. (In the example above this is the same as cut 5.)

→Robert: pointed out that cut 5 is incorrect on the slides after a demonstration of the derivation.

3. *First UIP scheme*: Choose the cut closest to the conflict. This has been empirically shown to be the best [Zhang+ 2001].
4. *Choose asserting clauses*: Asserting clauses are those that only have one variable assigned at the current level. After backtracking they will become a unit clause.

3.2 Miscellaneous Details

Learned Clause deletion: Learned clauses can use up memory so there are several heuristics for removing them.

Restarts: After a restart, the learned clauses can be kept and can help guide the solver to a faster solution.

4. Comparing SAT and CP

Pingyu drew a synthesis comparing SAT and CP, as disciplines, from the following perspectives:

1. Methodology
2. Application
3. Architecture
4. Evolution

→Pingyu: The authors dismiss SAT's local search, but several of the most successful SAT solvers use local search.

→Professor: SAT struggles with what that CP handles well?

→Wesley: Real domains

→Professor: SAT also handles when fragments of the problem are 1st order logic or something more difficult, in what is know SAT Modulo Theory (SMT).