Assignment 1

CSC2512—Winter 2020

Out: Feb 11th, 2020
Due: Monday March 2nd (email me your write-up)

1 Questions

Q1. 25/100 Consider the following CNF. (x_1, x_2), $(x_1, \neg x_2, x_4)$, $(\neg x_4, x_3, x_5, \neg x_6)$, $(\neg x_4, x_3, x_5, x_6)$, $(x_3, \neg x_5, x_7)$, $(x_3, \neg x_5, \neg x_7)$, $(\neg x_3, \neg x_4, x_8)$, $(\neg x_3, \neg x_4, \neg x_8)$, $(\neg x_1, \neg x_2)$, $(\neg x_1, x_2, \neg x_3)$, $(x_2, x_3, \neg x_4)$, $(\neg x_1, x_4, x_8)$, $(\neg x_1 x_4, \neg x_8)$

Show how CDCL would solve this formula when it always branches on the variable with the lowest index among the set of uninstantiated variables (x_i before x_j if $i < j$). The search always sets each decision variable to true.

To show CDCL's operation, draw the sequence of full configurations of the trail the algorithm explores, where each full configuration is the trail extended until it reaches a conflict or assigns all variables.

Write each full trail as a sequence of literals made true in the trail, marking each literal as being a decision or a unit implied literal. For each unit implied literal show the reason clause. If the trail results in a conflict, show the conflict clause, and *show the resultant 1-UIP clause*.

Use the 1-UIP clause (if one is found) to generate the next trail configuration by backtracking the previous configuration, adding the literal implied by the 1-UIP clause and then continuing down until once again you have fully extended the trail. *Assume that all learnt 1-UIP clauses are remembered and used when generating the subsequent trail configurations.*

Finally, show the *resolution proof* that CDCL has found (as a sequence of clauses).

Q2. 75/100 Cardinality constraints are an important type of constraint that appear in many SAT problems. These are constraints of the form $\sum_{x \in S} x \leq k$. That is, of the Boolean *literals* in the set S at most k of them can be assigned the value TRUE. (The sum assumes that TRUE is equal to 1, while FALSE is equal to 0). For example, $x_1 + \neg x_2 + x_3 \leq 1$ is a cardinality constraint that states that of the literals x_1, $\neg x_2$ and x_3 at most one can be true.

In the paper "Efficient CNF Encoding of Boolean Cardinality Constraints" by Bailleux and Boufkhad you will find a description of one way of encoding this kind of constraint into SAT.

Implement this encoding. You can choose any programming language you want to do your implementation. SAT solvers take their input as a file of clauses specified in DIMACS format. In DIMACS format, literals are represented as positive and negative integers (not equal to 0). So your implementation should take as input

1. The set S specified as a sequence of positive or negative integers (representing the literals in the sum).
2. The number k that is the upper bound of the sum.
3. A number n which specifies where the new variables introduced by the encoding are to start.

That is, all of the auxiliary variables (i.e., new variables) used in your encoding are to be numbered from n upwards.

Your implementation should produce as output a sequence of clauses that are only satisfied by truth assignments in which the number of true literals in S is less than or equal to k.

Using your implementation, produce a DIMACS formatted file with the clausal encoding of the cardinality constraint, and test the encoding with MiniSat (which you can download at `https://github.com/niklasso/minisat`). The solutions produced by MiniSat should satisfy the cardinality constraint. You can experiment with multiple cardinality constraints in one file and or with cardinality constraints of different sizes.

Do some experiments (10-20 test cases should suffice) to test how your encoding performs as the number of literals in S, the bound k, and the number of different cardinality constraints varies. Describe the experiments and draw some conclusions. (Note that in most cases a SAT problem encoding just one cardinality constraint is likely to be very quickly solved by MiniSat, but if you add multiple different cardinality constraints over intersecting sets S the problem can quickly become hard.)

In your hand in, describe what experiments you ran to test the performance of your encodings. Give some conclusions drawn from your experience in implementing and testing your encoding. **Keep your write up short, maximum of 3 pages for this last question.**

You might also find the paper "SAT Encodings of the At-Most-k Constraint" by Bittner, Thm, and Schaefer interesting.

2 Notes

See the web page for links to MiniSat, documentation about the DIMACs format and the papers mentioned.