CS 466/666 Spring 2013

Special cases of NP-Complete Problems: CNF, 3-Sat and 2-SAT

We have proved that SAT is NP-Complete. Indeed, if you look at the proof, you see that only a special case of SAT is used, namely the product (and) of sums (or) of literals (variables and their negations). This $\Pi\ \Sigma$ (literals) form is called **conjunctive normal form.** (Similarly, the $\Sigma\ \Pi$ (literals) form is called **conjunctive normal form**) One can convert an arbitrary Boolean formula to either of these forms, but it may result in a dramatic (exponential) increase in size. The first question we ask, though, is how much more we can restrict the form of a Boolean expression and still have its Sat problem NP-complete. Curiously, 3 literals per clause is as hard as the general problem. We can reduce general CNF-SAT to CNF-3SAT (which we will simply call 3-SAT). Consider an arbitrary CNF expression $\Pi\ C_i$, where $C_i = (a_{i1} \vee a_{i2} \vee \ \dots\ a_{ik})$ and a_{ij} denotes a literal. The reduction is quite straightforward and causes only a linear increase in size.

Write a clause $C = (a_1 \vee a_2 \vee \ \dots\ a_k)$ as $(a_1 \vee a_2 \vee b_1)\ \Pi_{i=1,..k-4}\ (\neg b_i \vee a_{i+2} \vee b_{i+1})\ (\neg b_{k-3} \vee a_{k-1} \vee a_k)$. Here the b's are new variables not used anywhere else in the construction. If C is made true by assigning a_j true, then the new form is made true by making a_j true, all b_i that occur earlier than the clause where a_j are made true and all that come later than this clause are made false.

So how about just 2 literals per clause? Things become dramatically easier:

2-Sat is easily solved in polynomial time. Simply make an arbitrary assignment of one variable. This satisfies some clauses and forces the assignment of other variables if certain other clauses are to be satisfied. The process of forced assignment and clause elimination continues until we either eliminated some clauses and no remaining variables are forced, or we have a contradiction (the same variable is forced to be true in one clause and false in another). In the former case, we keep the assignments and repeat the process until no clauses are remain unsatisfied. In the latter case, our tentative assignment does not work, so we must make the opposite one … which goes through a similar forcing process and leads either to the conclusion that the expression is not satisfiable or finds a satisfying assignment. It is not hard to eliminate a single variable in O(n), but indeed then entire process can be done in linear time.

Input: CNF expression of n clauses each contains 2 literals
(A literal is a Boolean variable or its negative)

Output: A satisfying assignment of variables or, essentially, a proof if these are none.

Note: There are most 2n variables

Data Structures:
For each variable x_i create
doubly linked list of clauses containing x_i (without $\neg$) and
doubly linked list of clauses containing $\neg\ x_i$.

Each clause representation has 2 sets of flags to give status of each variable (T, F,?) and the clauses (T,?)

Algorithm:
Choose an arbitrary variable x_i from a clause still unsatisfied. Simultaneously (e.g. you could alternate steps) run 2 processes one makes tentative assignment x_i =T, the other makes x_i =F, (Data structures marked independently by the two)

All clauses made true by the assignment are flagged as such. For all clauses in which the negation of the assigned value occurs, the other literal of that clause is forced to hold. Continue with these forced assignments (successively) until either:

i) All assignments have been made; hence every clause is satisfied or has no assignment to either term.

or

ii) We discover some variable is forced to both T & F. Hence a contradiction to original assignment.

Case (i): Halt the other process and run through it again undoing all its markings. (This takes same amount of time as making assignment). Assignments made by the properly terminating process are "made permanent". This leaves us with a subset of clauses & variables, all these remaining clauses in original form. The time taken is the same for each process; the number of clauses removed is proportional to time taken, as a clause is inspected at most twice by properly termination process. Reapply procedure to remaining clauses starting with some remaining variables (dual) assignment.

Case (ii) Terminating assignment leads to contradiction. Let other assignment process continue to completion. Satisfaction if it finds one.