Satisfiability

Basics

A *Boolean variable* takes the value true or false. Using the *and*, *or*, and *not* operators, we can form Boolean expressions whose truth value depends on the truth value of the variables in the expression. For example (a *or not* b) *and* (a *or* b) is true precisely if a is true.

We use $\overline{x}$ as shorthand for *not* x. We use $\wedge$ for *and* and $\vee$ for *or*. A *literal* is either a Boolean variable or its negation. Thus, we write the above expression as

$$(a \vee \overline{b}) \wedge (a \vee b)$$

using the three literals a, b, and $\overline{b}$.

We shall consider expressions of the form $C_1 \wedge C_2 \wedge .. \wedge C_l$ where each C_i consists of a sequence of literals with $\vee$ between them. These formulas are said to be in *Conjuctive Normal Form.*

For example

$$(a \vee b) \wedge (c \vee \overline{d}) \wedge (\overline{a} \vee b \vee \overline{c})$$

is a CNF formula whilst

$$(a \wedge b) \vee (a \vee d)$$

is not.

We refer to the C_i as the clauses of the CNF formula. We will express a CNF formula simply as a list of clauses without explicitly writing the $\wedge$ symbols between them. Furthermore, we give each clause as a list of literals without explicitly writing the $\vee$ symbols between them.

We note that for a given choice of truth values for the variables within it, a CNF formula evaluates to true precisely if each clause within it evaluates to true. We say that a truth assignment *satisfies* a CNF formula if with this truth assignment, the formula evaluates to true.

The *Satisfiability* problem has as input a CNF formula with clauses $C_1, ..., C_l$ using variables $x_1, ..., x_n$. The problem is to find a satisfying truth assignment for it or determine that no such assignment exists.

Some Examples

Many problems can be reduced to Satisfiability. For example, suppose we want to determine if we can partition the vertices of a graph up into two sets C and S such that every pair of vertices of C are joined by an edge and no pair of vertices of S is joined by an edge. In standard terminology, C is a **c**lique and S is a **s**table set. We define the corresponding instance of Satisfiability as follows. We have a variable x_v for each vertex of the graph which corresponds to the statement: v is in the clique. For each edge uv of the graph, we cannot put both u and v in S so we have the clause (x_u, x_v). For each pair (u, v) of non-adjacent vertices, we cannot put both u and v in C so we have the clause

$(\overline{x_u}, \overline{x_v})$. Clearly, given a partition of G into a clique C and stable set S setting x_v to true if v is in C and to false if v is in S yields a satisfying truth assignment. Conversely given a satisfying truth assignment, letting C be the set of those v such that x_v is true and S be those for which x_v is false yields a partition of G into a clique and a stable set. So solving one problem solves the other.

As a more interesting example, suppose we want to determine if we can schedule exams for courses $A_1, ..., A_p$ in Rooms $R_1, ..., R_r$ at times $T_1, ..., T_s$. We need to avoid scheduling conflicts for the exams for pairs of courses which are both taken by many students. Also, some rooms are too small for certain exams. We assume we are given a list $\mathcal{L}$ of pairs (A_i, A_j) of conflicts (i.e the exams for these courses must be at different times) and a list $\mathcal{J}$ of all the course-room pairs (A_i, R_j) such that R_j is too small for the exam for A_i.

We define a CNF formula on the variables $X_{i,j,k}$ for $1 \leq i \leq p, 1 \leq j \leq r, 1 \leq k \leq s$. where $X_{i,j,k}$ corresponds to the statement: the exam for A_i is being held in room R_j at time T_k.

To make sure that there is an exam for course A_i for each i between 1 and p we have the clause

$$\{X_{i,j,k} | 1 \leq j \leq r, 1 \leq k \leq s\}$$

To make sure that there is only one exam for course i, for each i and all distinct pairs $(j,k), (j',k')$ with $1 \leq j, j' \leq r$ and $1 \leq k, k' \leq s$ we have the clause:

$$\overline{X_{i,j,k}}, \overline{X_{i,j',k'}}.$$

To make sure no two exams are scheduled at the same place at the same time, for $1 \leq i, i' \leq p$, and $1 \leq j \leq r, 1 \leq k \leq s$, we have the clause:

$$\overline{X_{i,j,k}}, \overline{X_{i',j,k}}.$$

To avoid conflicts for every pair (A_i, A_i') on $\mathcal{L}$ and every j, j', k with $1 \leq j, j' \leq r, 1 \leq k \leq s$ we have the clause:

$$\overline{X_{i,j,k}}, \overline{X_{i',j',k}}.$$

To avoid overcrowding, for each pair (A_i, R_j) on $\mathcal{J}$ and each k between 1 and s, we have the clause:

$$\overline{X_{i,j,k}}.$$

A satisfying truth assignment points out a feasible schedule and conversely (with $X_{i,j,k}$ being true precisely if course A_i is scheduled in room R_j at time T_k).

k-SAT

An instance of $k-SAT$ is an instance of SAT in which each clause contains at most one literal. Thus, our first example in the last section was an instance of 2-SAT.

An instance of 1-SAT is satisfiable if and only if there is no variable x such that both x and $\overline{x}$ are clauses.

As we saw in class, instances of 2-SAT with n variables and m clauses can be solved efficiently using depth first search.

No efficient algorithm for 3-SAT is known and indeed it is widely believed that no such algorithm exists. Such an algorithm could be used to solve the general SAT problem and many other notoriously difficult problems. Determining if there is such an algorithm would answer the million dollar question:

$$P = NP?$$