36.4-5 Give a polynomial-time algorithm to satisfy Boolean formulas in disjunctive normal form.

Satisfying one clause in DFS satisfied the whole formula. One clause can always be satisfied iff it does not contain both a variable and its complement.

Why not use this reduction to give a polynomial-time algorithm for 3-SAT? The DNF formula can become exponentially large and hence the reduction cannot be done in polynomial time.

A Perpetual Point of Confusion

Note carefully the direction of the reduction.

We must transform *every* instance of a known NP-complete problem to an instance of the problem we are interested in. If we do the reduction the other way, all we get is a slow way to solve x, by using a subroutine which probably will take exponential time.

This always is confusing at first - it seems bass-ackwards. Make sure you understand the direction of reduction now - and think back to this when you get confused.

Integer Programming

Instance: A set v of integer variables, a set of inequalities over these variables, a function $f(v)$ to maximize, and integer B.

Question: Does there exist an assignment of integers to v such that all inequalities are true and $f(v) \geq B$?

Example:

$$v_1 \geq 1, \quad v_2 \geq 0$$

$$v_1 + v_2 \leq 3$$

$$f(v) : 2v_2, \quad B = 3$$

A solution to this is $v_1 = 1$, $v_2 = 2$.

Example:

$$v_1 \geq 1, \quad v_2 \geq 0$$

$$v_1 + v_2 \leq 3$$

$$f(v) : 2v_2, \quad B = 5$$

Since the maximum value of $f(v)$ given the constraints is $2 \times 2 = 4$, there is no solution.

Theorem: Integer Programming is NP-Hard

Proof: By reduction from Satisfiability

Any set instance has boolean variables and clauses. Our Integer programming problem will have twice as many variables as the SAT instance, one for each variable and its compliment, as well as the following inequalities:

For each variable v_i in the set problem, we will add the following constraints:

- $1 \leq V_i \leq 0$ and $1 \leq \overline{V}_i \leq 0$

  Both IP variables are restricted to values of 0 or 1, which makes them equivalent to boolean variables restricted to true/false.

- $1 \leq V_i + \overline{V}_i \leq 1$

  Exactly one of the IP variables associated with a given sat variable is 1. This means that exactly one of V_i and $\overline{V}_i$ are true!

- for each clause $C_i = \{v_1, \overline{v}_2, \overline{v}_3 \ldots v_n\}$ in the sat instance, construct a constraint:

$$v_1 + \overline{v}_2 + \overline{v}_3 + \ldots v_n \geq 1$$

  Thus at least one IP variable must be one in each clause! Thus satisfying the constraint is equivalent to satisfying the clause!

Our maximization function and bound are relatively unimportant: $f(v) = V_1$ $B = 0$.

Clearly this reduction can be done in polynomial time.

We must show:

1. Any SAT solution gives a solution to the IP problem.

   In any SAT solution, a TRUE literal corresponds to a 1 in the IP, since if the expression is SATISFIED, at least one literal per clause in TRUE, so the sum in the inequality is ≥ 1.

2. Any IP solution gives a SAT solution.

   Given a solution to this IP instance, all variables will be 0 or 1. Set the literals correspondly to 1 variable TRUE and the 0 to FALSE. No boolean variable and its complement will both be true, so it is a legal assignment with also must satisfy the clauses.

Neat, sweet, and NP-complete!

Things to Notice

1. The reduction preserved the structure of the problem. Note that the reduction did not *solve* the problem – it just put it in a different format.

2. The possible IP instances which result are a small subset of the possible IP instances, but since some of them are hard, the problem in general must be hard.

3. The transformation captures the essence of why IP is hard - it has nothing to do with big coefficients or big ranges on variables; for restricting to 0/1 is enough. A careful study of what properties we do need for our reduction tells us a lot about the problem.

4. It is not obvious that IP $\leq$ NP, since the numbers assigned to the variables may be too large to write in polynomial time - don't be too hasty!