Lecture 3: Solving 2-SAT and Max-Flow

We recalled the definition of the Satisfiability problem, an instance of which consists of a set C of Boolean clauses over a set X of Boolean variables. We noted that an instance of 2-SAT was an instance of Satisfiability in which every clause has size 2.

We noted that the clause a OR b for two literals a and b was equivalent to the implication NOT A => B and to the implication NOT B=>A. To each instance (C,X) of 2-SAT we associated a digraph G=(V,E) where V={x, NOT x |x is in X}, and E is the set of directed edges xy, for every pair of literals such that NOT x OR y is a clause.

W remarked that we have a satisfying truth assignment precisely if we can choose a set T of vertices of G containing exactly one of each pair (x, Not x) so that there is no (directed) edge xy with x in T and y not in T. Thus we have reduced our instance of 2-SAT to determining if such a choice exists. We note that for any such assignment, for every (directed) path of G, if the first vertex on the path is in T, then all the vertices in the path are in T (we can prove this by induction on the length of the path). This implies that for every strong component K of G, either all of its vertices are in T or all of its vertices are not in T.

Thus, there is no such assignment if some variable x lies in the same strong component as its negation.

We also noted that because we add edges in pairs xy is an edge precisely if there is an edge from NOT y to NOT x. Applying induction on the length of a path from x to y, we see that there is a path from x to y, precisely if there is a path from NOT y to NOT x.

Applying this twice,we see that there is a path from x to y and a path from y to x precisely if there is a path from NOT y to NOT x and a path from NOT x to NOT y. That is, the set of literals in the same strong component as NOT x are precisely the negation of the literals in the the same strongly connected component as x.

Our algorithm for solving the problem to which we have reduced 2-SAT proceeds as follows. We first determine the strongly connected components of G (this can be done in O(|V|+|E|) time, as set out in CLRS pp. 615-620 and discussed in COMP252). We can assign a distinct integer to each strongly connected component and build in O(|V|) time, an array indexed by the vertices, so that for vertex we record the number of the component it is in. Using the array, we can check in O(|V|) time if there is a variable x, such that x and NOT x are in the same strongly connected component. If so, we return that we cannot choose a T as desired.

Otherwise, we build the component graph of G (which I called a reduction in the lecture) whose vertices are the integers assigned to its strong components and where there is an edge from the integer assigned to J to the integer assigned to K, if in G, there is an edge from a vertex in J to a vertex in K. As shown in CLRS p. 617 this reduction can easily be constructed in linear time and is acyclic.

We find a topological sort of the reduction which yields an ordering on its vertices. We consider the integers corresponding to the strong components, and the corresponding strong components, in the reverse of this topological order.
We build an array CHOICE indexed by the vertices where CHOICE[x] is TRUE if we have put x in T and FALSE otherwise. We initialize this array to contain only FALSE. We actually implement CHOICE as a doubly indexed array indexed by the boolean variables of our 2-SAT instance and a second index which is (unnegated or negated). Thus given a literal we can access CHOICE for its negation in constant time.
For each strong component K in turn, we put all the vertices of K in T, and set CHOICE[x] to be be true for ach such vertex unless CHOICE[NOT x] is true for some (and hence all) the vertices x in K. We can do this in time proportional in the size of K, and hence in O(|V|) time in total. It follows that the total time taken for an instance of 2-SAT with variables X and set of Clauses C is O(|X|+|C|).
To see that this process does not create an edge xy of G where x is in T but y is not, consider such an edge. Then x is in some strong component J of G and y is in some strong component K of G, and xy yields an edge of the reduction from the integer j corresponding to J and the integer k corresponding to K. Now, J is paired with a strong component J' which contains the negation of the literals within it, and K is paired with a strong component K' which contains the negation of the literals within it. Furthermore, because of the way our edges come in pairs, not y not x is an edge of G, which corresponds to an edge of the reduction from the integer k' corresponding to K' to the integer j' corresponding to J'. But in the topological sort, we have that k' comes after k as y is not in T. We also have j comes before k because of the edge jk. Finally we have that j' comes before j because x is in T. So, j' comes before k' contradicting the fact that j'k' is an edge.
This completes the proof of correctness of our algorithm for 2-SAT, which as we have seen runs in linear time.

We then presented the augmenting path algorithm for Max-Flow discussed in Chvatal pp. 374-380. We noted that it proved the Max-Flow Min Cut Theorem. We also noted that it proved that if all capacities were integer or infinite then the optimal solution would be integer(or infinite). We noted that if all the capacities were integer and at most C then the number of iterations was at most C|E|. We gave an example that showed that if there were infinite capacity edges then if we did not choose the augmenting paths carefully then the number of iterations could be infinite. We noted (but did not prove) that if we always augmented along a shortst path of the auxiliary network then the number of iterations was at most |V||E|/2 regardless of the capacities. Those interested can read the proof in Chvatal.