Notes by Samir Khuller.

26 NP-Completeness

In the last lecture we showed every non-deterministic Turing Machine computation can be encoded as a SATISFIABILITY instance. In this lecture we will assume that every formula can be written in a restricted form (3-CNF). In fact, the form is $C_1 \cap C_2 \cap \ldots \cap C_m$ where each C_i is a "clause" that is the disjunction of exactly three "literals". (A literal is either a variable or its negation.) The proof of the conversion of an arbitrary boolean formula to one in this form is given in [5].

The reductions we will study today are:

1. SAT to CLIQUE.
2. CLIQUE to INDEPENDENT SET.
3. INDEPENDENT SET to VERTEX COVER.
4. VERTEX COVER to DOMINATING SET.
5. SAT to DISJOINT CONNECTING PATHS.

The first three reductions are taken from Chapter 36.5 [5] pp. 946–949.

Vertex Cover Problem: Given a graph $G = (V, E)$ and an integer K, does G contain a vertex cover of size at most K? A vertex cover is a subset $S \subseteq V$ such that for each edge (u, v) either $u \in S$ or $v \in S$ (or both).

Dominating Set Problem: Given a graph $G' = (V', E')$ and an integer K', does G' contain a dominating set of size at most K'? A dominating set is a subset $S \subseteq V$ such that each vertex is either in S or has a neighbor in S.

DOMINATING SET PROBLEM:

We prove that the dominating set problem is NP-complete by reducing vertex cover to it. To prove that dominating set is in NP, we need to prove that given a set S, we can verify that it is a dominating set in polynomial time. This is easy to do, since we can check membership of each vertex in S, or of one of its neighbors in S easily.

Reduction: Given a graph G (assume that G is connected), we replace each edge of G by a triangle to create graph G'. We set $K' = K$. More formally, $G' = (V', E')$ where $V' = V \cup V_e$ where $V_e = \{v_{e_i} | e_i \in E\}$ and $E' = E \cup E_e$ where $E_e = \{(v_{e_i}, v_k), (v_{e_i}, v_\ell) | e_i = (v_k, v_\ell) \in E\}$. (Another way to view the transformation is to subdivide each edge (u, v) by the addition of a vertex, and to also add an edge directly from u to v.)

If G has a vertex cover S of size K then the same set of vertices forms a dominating set in G'; since each vertex v has at least one edge (v, u) incident on it, and u must be in the cover if v isnt. Since v is still adjacent to u, it has a neighbor in S.

For the reverse direction, assume that G' has a dominating set of size K'. Without loss of generality we may assume that the dominating set only picks vertices from the set V. To see this, observe that if v_{e_i} is ever picked in the dominating set, then we can replace it by either v_k or v_ℓ, without increasing its size. We now claim that this set of K' vertices form a vertex cover. For each edge e_i, v_{e_i}) must have a neighbor (either v_k or v_ℓ) in the dominating set. This vertex will cover the edge e_i, and thus the dominating set in G' is a vertex cover in G.

DISJOINT CONNECTING PATHS:

Given a graph G and k pairs of vertices (s_i, t_i), are there k vertex disjoint paths $P_1, P_2, \ldots, P_k$ such that P_i connects s_i with t_i ?

This problem is NP-complete as can be seen by a reduction from 3SAT.

Corresponding to each variable x_i, we have a pair of vertices s_i, t_i with two internally vertex disjoint paths of length m connecting them (where m is the number of clauses). One path is called the *true* path

and the other is the *false* path. We can go from s_i to t_i on either path, and that corresponds to setting x_i to true or false respectively.

For clause $C_j = (x_i \wedge \overline{x_\ell} \wedge x_k)$ we have a pair of vertices s_{n+j}, t_{n+j}. This pair of vertices is connected as follows: we add an edge from s_{n+j} to a node on the false path of x_i, and an edge from this node to t_{n+j}. Since if x_i is true, the s_i, t_i path will use the true path, leaving the false path free that will let s_{n+j} reach t_{n+j}. We add an edge from s_{n+j} to a node on the true path of x_ℓ, and an edge from this node to t_{n+j}. Since if x_ℓ is false, the s_ℓ, t_ℓ path will use the false path, leaving the true path free that will let s_{n+j} reach t_{n+j}. If x_i is true, and x_ℓ is false then we can connect s_{n+j} to t_{n+j} through either node. If clause C_j and clause $C_{j'}$ both use x_i then care is taken to connect the s_{n+j} vertex to a distinct node from the vertex $s_{n+j'}$ is connected to, on the false path of x_i. So if x_i is indeed true then the true path from s_i to t_i is used, and that enables both the clauses C_j and $C_{j'}$ to be true, also enabling both s_{n+j} and $s_{n+j'}$ to be connected to their respective partners.

Proofs of this construction are left to the reader.

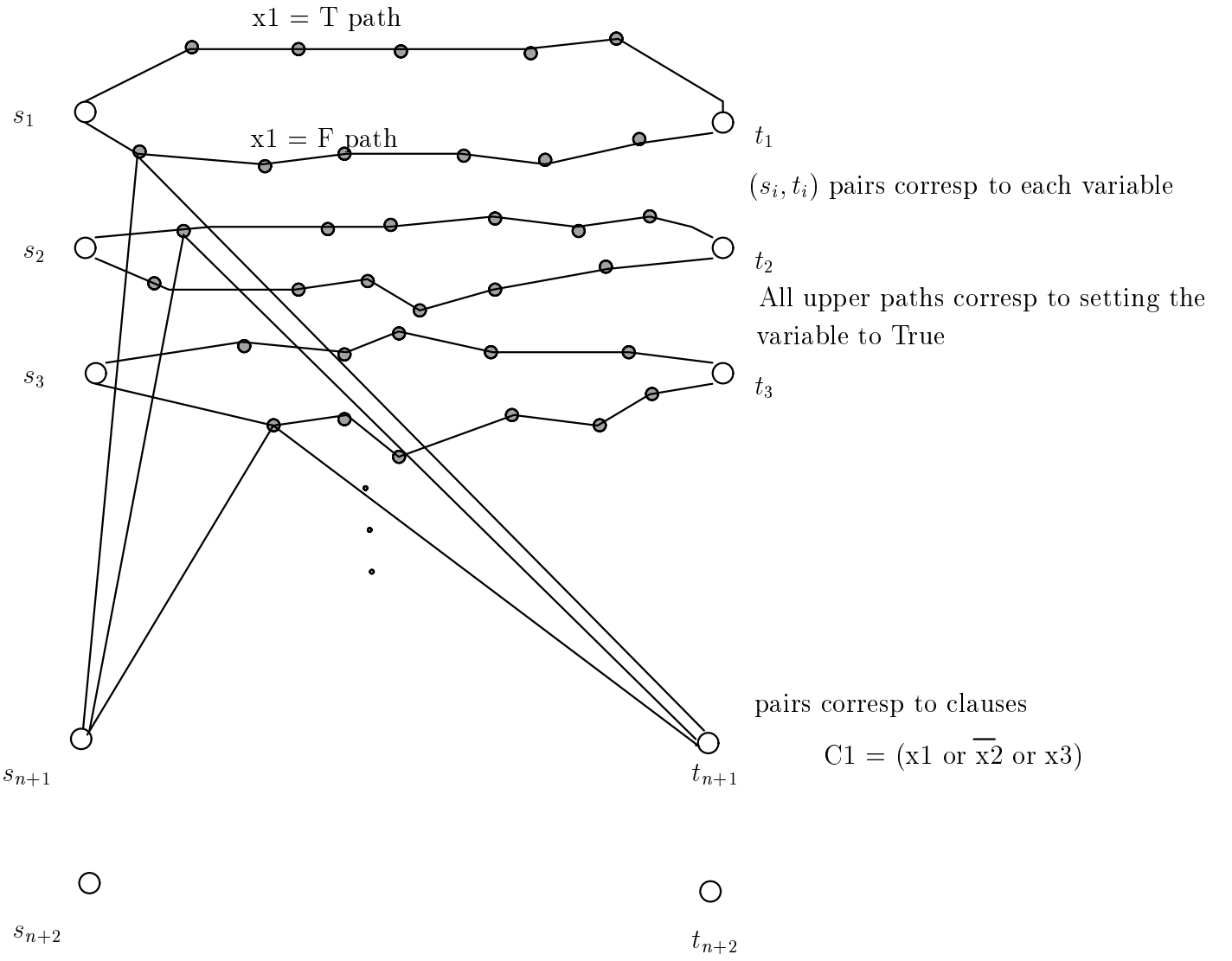

Figure 28: Graph corresponding to formula

Notes by Samir Khuller.

27 NP-Completeness

We will assume that everyone knows that SAT is NP-complete. The proof of Cook's theorem was given in CMSC 650, which most of you have taken before. The other students may read it from [CLR] or the book by Garey and Johnson.

The reduction we will study today is: CLIQUE to MULTIPROCESSOR SCHEDULING.
MULTIPROCESSOR SCHEDULING: Given a DAG representing precedence constraints, and a set of jobs J all of unit length. Is there a schedule that will schedule all the jobs on M parallel machines (all of the same speed) in T time units ?

Essentially, we can execute upto M jobs in each time unit, and we have to maintain all the precedence constraints and finish all the jobs by time T.
Reduction: Given a graph $G = (V, E)$ and an integer k we wish to produce a set of jobs J, a DAG as well as parameters M, T such that there will be a way to schedule the jobs if and only if G contains a clique on k vertices.

Let n, m be the number of vertices and edges in G respectively.

We are going to have a set of jobs $\{v_1, \ldots, v_n, e_1, \ldots, e_m\}$ corresponding to each vertex/edge. We put an edge from v_i to e_j (in the DAG) if e_j is incident on v_i in G. Hence all the vertices have to be scheduled before the edges they are incident to. We are going to set $T = 3$.

Intuitively, we want the k vertices that form the clique to be put in time slot 1. This makes $C(k, 2)$ edges available for time slot 2 along with the remaining $n - k$ vertices. The remaining edges $m - C(k, 2)$ go in slot 3. To ensure that no more than k vertices are picked in the first slot, we add more jobs. There will be jobs in 3 sets that are added, namely B, C, D. We make each job in B a prerequisite for each job in C, and each job in C a prerequisite for each job in D. Thus all the B jobs have to go in time 1, the C jobs in time 2, and the D jobs in time 3.

The sizes of the sets B, C, D are chosen as follows:

$$|B| + k = |C| + (n - k) + C(k, 2) = |D| + (m - C(k, 2)).$$

This ensures that there is no flexibility in choosing the schedule. We choose these in such a way, that $\min(|B|, |C|, |D|) = 1$.

It is trivial to show that the existence of a clique of size k implies the existence of a schedule of length 3. The proof in the other direction is left to the reader.

Notes by Samir Khuller.

28 More on NP-Completeness

The reductions we will study today are:

1. 3 SAT to 3 COLORING.
2. 3D-MATCHING (also called 3 EXACT COVER) to PARTITION.
3. PARTITION to SCHEDULING.

I will outline the proof for 3SAT to 3COLORING. The other proofs may be found in Garey and Johnson.

We are given a 3SAT formula with clauses $C_1, \ldots, C_m$ and n variables $x_1, \ldots, x_n$. We are going to construct a graph that is 3 colorable if and only if the formula is satisfiable.

We have a clique on three vertices. Each node has to get a different color. W.l.o.g, we can assume that the three vertices get the colors $T, F, *$. (T denotes "true" F denotes "false".)

For each variable x_i we have a pair of vertices X_i and $\overline{X_i}$ that are connected to each other by an edge and also connected to the vertex $*$. This forces these two vertices to be assigned colors different from each other, and in fact there are only two possible colorings for these two vertices. If X_i gets the color T, then $\overline{X_i}$ gets the color F and we say that variable x_i is true. Notice that each variable can choose a color with no interference from the other variables. These vertices are now connected to the gadgets that denote clauses, and ensure that each clause is true.

We now need to construct an "or" gate for each clause. We want this gadget to be 3 colorable if and only if one of the inputs to the gate is T (true). We construct a "gadget" for *each* clause in the formula. For simplicity, let us assume that each clause has exactly three literals.

If all three inputs to this gadget are F, then there is no way to color the vertices. The vertices below the input must all get color $*$. The vertices on the bottom line now cannot be colored. On the other hand if any input is T, then the vertex below the T input can be given color F and the vertex below it (on the bottom line) gets color $*$, and the graph can be three colored.

If the formula is satisfiable, it is really easy to use this assignmenmt to the variables to produce a three coloring. We color each vertex X_i with T if x_i is True, and with F otherwise. (The vertex $\overline{X_i}$ is colored appropriately.) We now know that each clause has a true input. This lets us color each OR gate (since when an input it true, we can color the gadget). On the other hand, if the graph has a three coloring we can use the coloring to obtain a satisfying assignment (verify that this is indeed the case).

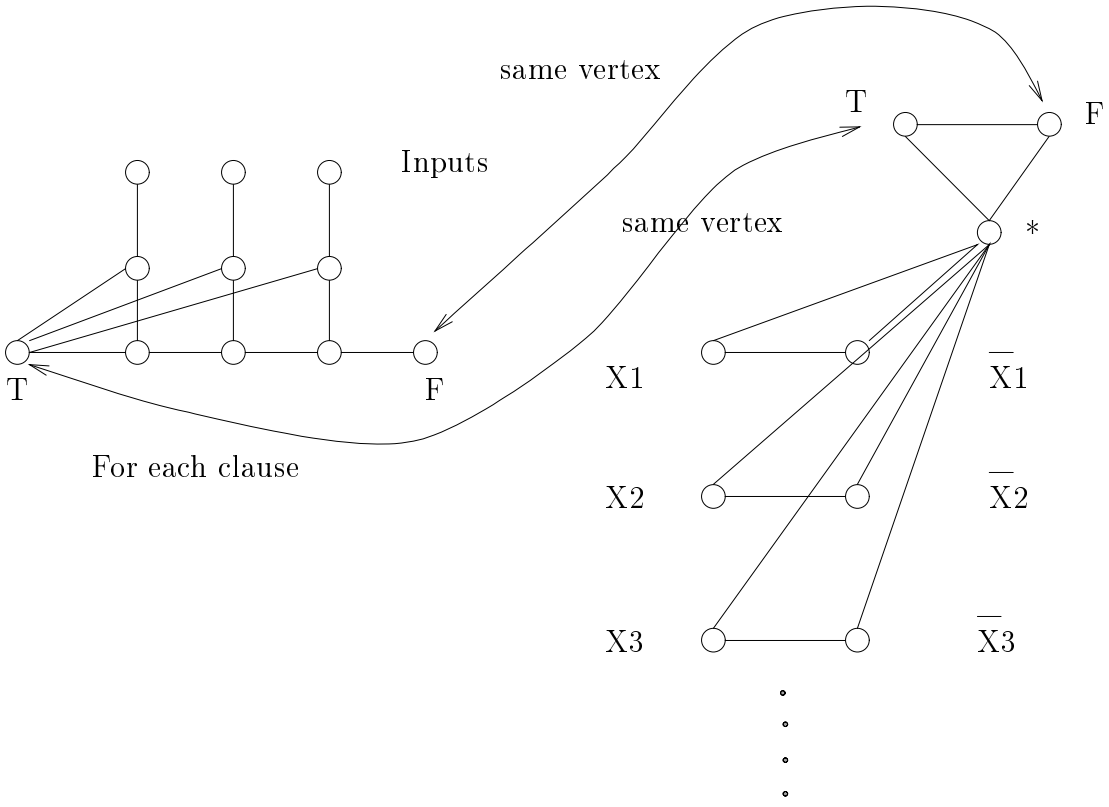

Figure 29: Gadget for OR gate