CSL 851: Algorithmic Graph Theory **Semester I 2013-2014**

Lecture 4: October 30

Lecturer: Amit Kumar *Scribes: Debjyoti Saharoy*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

4.1 Independent Set in Comparibility Graphs

We know that the size of any Independent Set is atmost the size of any Clique Cover (partioning the vertices of a Graph into cliques). So, the goal would be to find a clique cover and exhibit an Independent set of same size. Then it has to be the maximum Independent set.

4.1.1 The Construction

We assume that the orientation of the comparibility graph G is specified. For each vertex v of G we create two copies $x(v)$ (say the L side) and $y(v)$ (say the R side). For an edge $u \to v$ in G orient an edge from $x(u)$ to $y(v)$ having ∞ capacity. Also add a source s and a sink t. Direct edges from s to $x(v)$ $\forall v \in G$. Direct an edge from $y(v)$ to t $\forall v \in G$. The edges out of s and into t have capacity 1. Call this graph H. Let us demonstrate this using the following example.

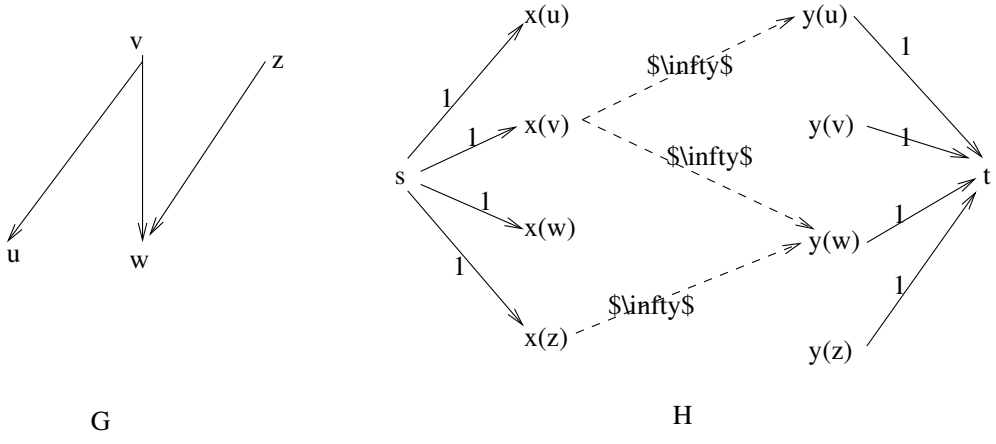


Figure 4.1: The comparibility graph G and the constructed graph H

4.1.2 Max flow in H

We will find max-flow ($= f_{max}$) in H. Since the edge capacities are integral so there exists a max-flow f_{max} such that the flow on each edge is either 0 or 1. If we consider the flow graph of H(the graph induced by the edges having flow 1 in the max-flow), the indegree and outdegree of all the vertices (except s and t) are atmost 1. Let us say the following is a max-flow graph of H. (For convenience we have named both the copies of vertices as v_i instead of $x(v_i)$ and $y(v_i)$.

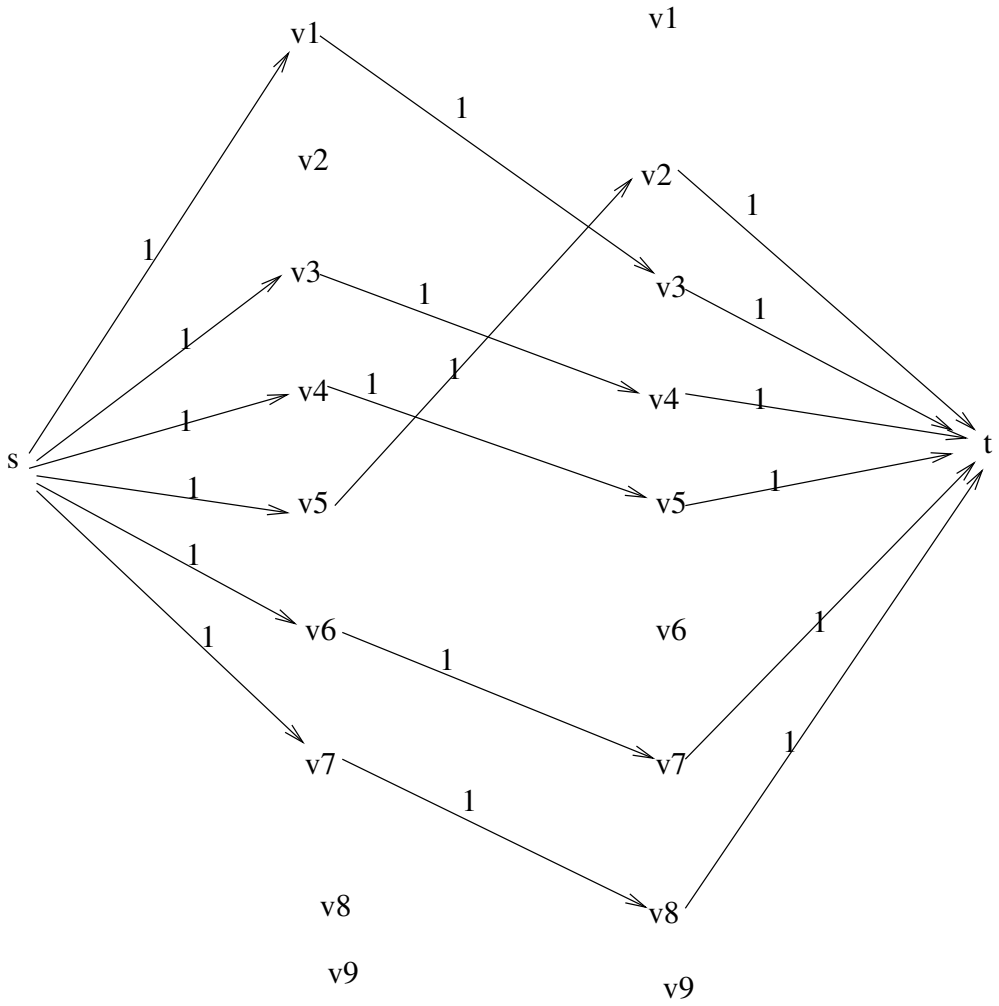
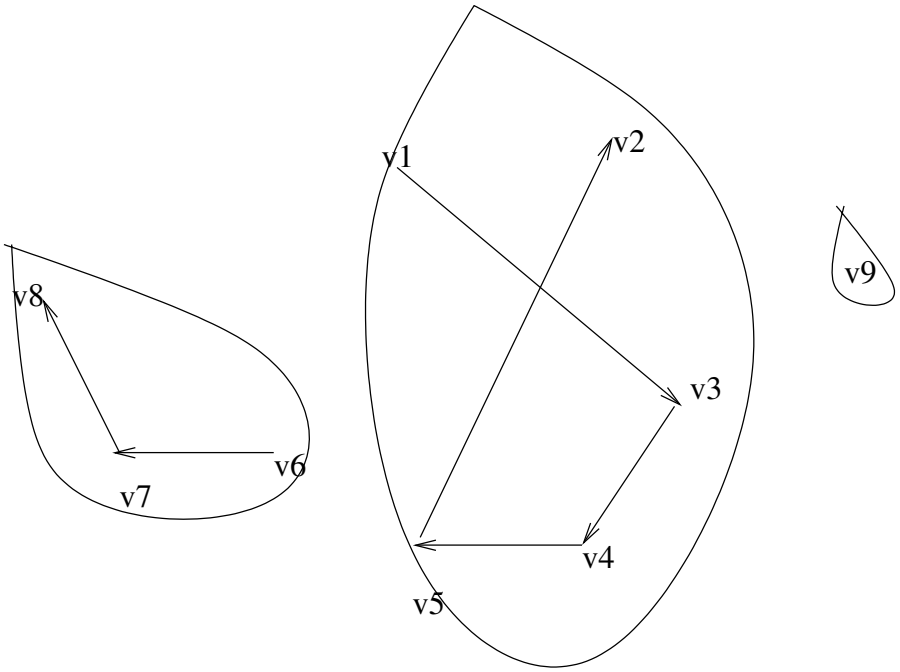


Figure 4.2: The edges having flow 1 in the max-flow graph of H

Let us now consider the graph in G induced by the edges which have flow 1 (and from L to R) in above max-flow graph. Since the indegree and outdegree of each vertex is atmost 1, so we will end up getting disjoint paths and may be also some lone vertices. This will look like the following,



We already argued in the previous class that paths in comparibility graphs are cliques. So what we have done here is partioned the graph G into cliques. The number of cliques = number of paths = number of vertices on the L side of max-flow graph of H which do not receive any flow from s (as those are the vertices where a path would end.) = n - f_{max}, where total number of vertices in G =n.

4.1.3 Constructing the Independent Set

Remember, we found the $s-t$ max-flow in the graph H (that we constructed from G). But, how would the $s-t$ min-cut in H look like ?? Let us say that S and $T = V - S$ be the sets which corresponds to the $s-t$ min-cut in H.

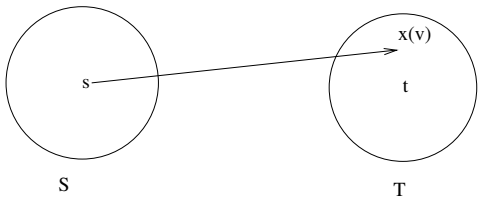


Figure 4.3: These type of edges would contribute +1 to the cut.

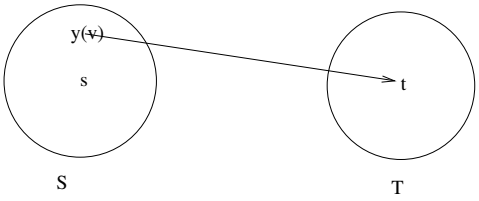


Figure 4.4: These type of edges would contribute +1 to the cut.

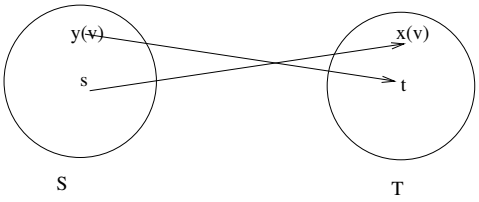


Figure 4.5: These type of edges would contribute +2 to the cut.

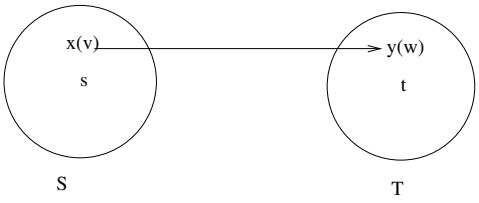


Figure 4.6: These type of edges can't be in the cut due to infinite capacity.

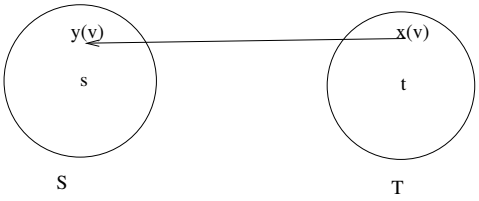


Figure 4.7: These type of edges possible but does not contribute anything to the cut

Let us define the set I as $I = \{v \in G : x(v) \in S \text{ and } y(v) \in T\}$
Claim 1: *min-cut* $\geq n - |I|$
The size of the set I^c is $n - |I|$ and each vertex in the set I^c contributes atleast 1 to the min-cut.Therefore, min-cut $\geq n - |I|$.
Claim 2: *I is an independent set.*
This is because had there been an edge between any pair of the vertices in I, it would make the min-cut have infinite capacity which is not possible.

So by invoking max-flow min-cut theorem,
n-Clique Cover $\geq n - |I|$
i.e Clique Cover $\leq |I|$.
But we also know, $|I| \leq$ Clique Cover. So $|I|$ = Clique Cover. Since we have found an Independent Set and a Clique Cover which are equal in size, we can claim that this better be the Maximum Independent set.

4.2 Independent Set in a Tree

Since trees(devoid of any cycles) are special chordal graphs(doesn't have induced cycle of length ≥ 4), to find Independent set we can use the algorithm (reverse-scan-greedy) that we studied in Lecture 2. But lets try dynamic programming.

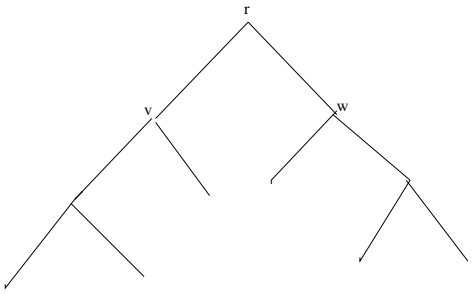


Figure 4.8: A tree

Let's define the d.p as follows: $I^1(x)$: value of the maximum independent set in the subtree below x provided x is in the independent set. $I^2(x)$: value of the maximum independent set in the subtree below x provided x is not in the independent set. $I(x) = max(I^1(x), I^2(x))$.
$I^1(r) = 1 + I^2(v) + I^2(w)$
$I^2(r) = I(v) + I(w)$
$I(r) = max(I^1(r), I^2(r))$

The question is for what other graphs can we formulate similar kind of d.p. The answer is tree-like graphs, this is what we discuss in the nect section.

4.3 Tree Decompositions and Tree-widths

The key idea behind formulating the d.p was as follows: The ″intersection″ region had a vertex whose removal disconnected the tree into two disjoint parts. Now if the number of vertices in the "intersection"

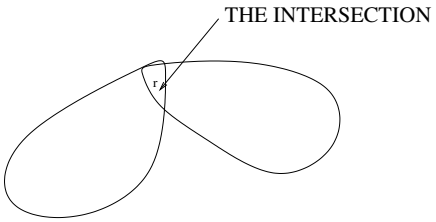


Figure 4.9: The d.p formulation

region is small and it satisfies the above property, then may be the graph can be called tree-like. Now lets define this formally.
Definition 1: A tree T is a tree-decomposition of $G = (V, E)$ if every vertex $x \in T$ corresponds to a set of vertices $S_x \subseteq V$ in G. And the following should hold:

1. for each $e = (u, v) \in G$ there is a $x \in T$ s.t $u, v \in S_x$
2. for any $v \in V$ the set of $x \in T$ s.t $v \in S_x$ should form a connected component.

Definition 2: tree-width of a tree decomposition T $= [max_{x \in T}|S_x|] - 1$.
Definition 3: tree-width of a graph G is the minimum tree-width over all tree-decompositions.

A trivial tree decomposition of a graph G would be all the vertices in a single node. This will have a very high tree-width. We would be interested in graphs having low tree-width as they would have polynomial algorithm (using d.p) for several NP-HARD graph problems. Now let us look at a few examples of tree-decompositions.

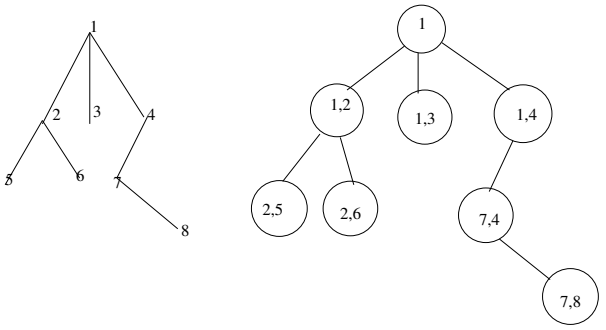


Figure 4.10: Tree decomposition of a tree. Trees have tree-width 1.

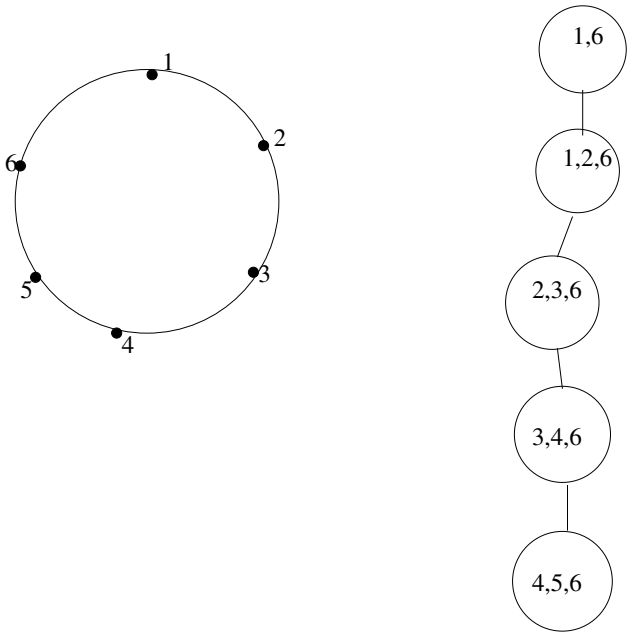


Figure 4.11: Tree decomposition of a cycle. Cycles have tree-width 2.

Theorem If p is a constant and tree-width of a graph G is atmost p, then there are polynomial time algorithms to find tree-decomposition of G of tree-width atmost p, otherwise it is NP-HARD and there exists constant time approximations.
(we did not prove this !)