Min Vertex Cover In Bipartite Graphs

Frank

November 26, 2021

1 Min Vertex Cover In Bipartite Graphs

1.1 Vertex Cover

- Suppose given $G = (V, E)$ undirected graph,

  Vertex Cover is $V' \subseteq V$ s.t. every edge in E has at least one endpoint in V'

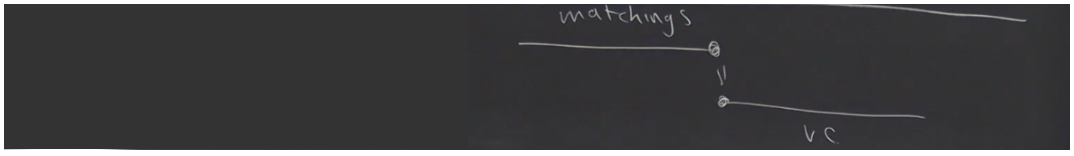
  Min V.C. is a vertex cover of min size

- For example, $\{b, d\}$ will be a minimum VC of the following graph

1.2 Problem

- Finding min v.c. is NP-hard in general, but poly-time for bipartite graphs
- **Claim 1:** For any graph G, $\forall$ matching M in G and $\forall$ v.c. $R \in G$, $|M| \leq |R|$
  - Argue by pigeon hole principle, if we have $|R| < |M|$, then M will not be a matching. Thus we must have $|M| \leq |R|$
- **Claim 2:** If M is a matching and R is a v.c. of G s.t. $|M| = |R| \implies M$ is max matching and R is min v.c.

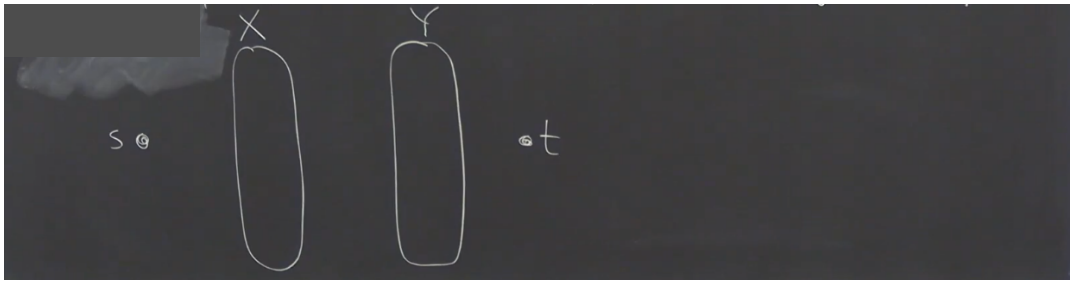
  Similar to max flow/min cut



- **Thm:** If $G = ((X, Y), E)$ is a bipartite graph, size of max matching in G = size of min v.c. in G (Not true in general for graphs)

  Proof:

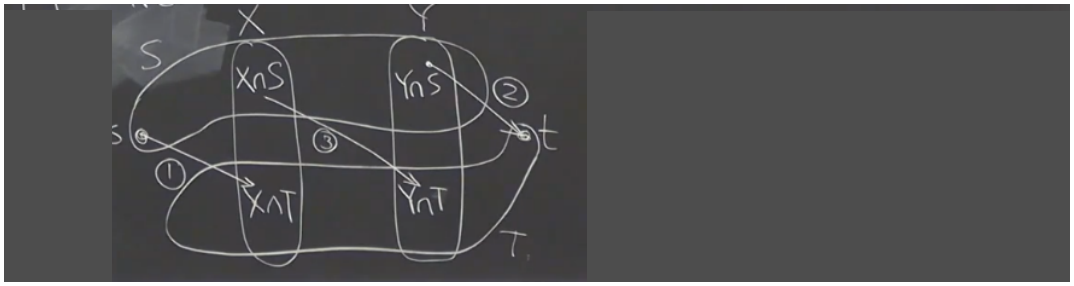
  - From G, construct flow network $F = (G', s, t, c)$ as before.



  - Let f be max flow in F
  - Construct residual graph G'_f, then G'_f does not have a path from s to t (because f is a max flow)
  - Now construct a cut of G'_f such that

    S = nodes reachable from s in G'_f

    T = rest of the nodes in G'
  - Now consider 3 different kinds of edges that goes from S to T



  - Will show that there are no edges of type 3 $\implies R = (X \cap T) \cup (Y \cap S)$ is a v.c.

    $|R|$ = number of edges of type 1 + number of edges of type 2

    $= c(S, T)$ [since no type 3 edges, and is constructed from max flow]

    $= V(f)$ [by max flow/min cut]

    = size of max matching in G [previous lecture]

  then by claim 2, R is min v.c.

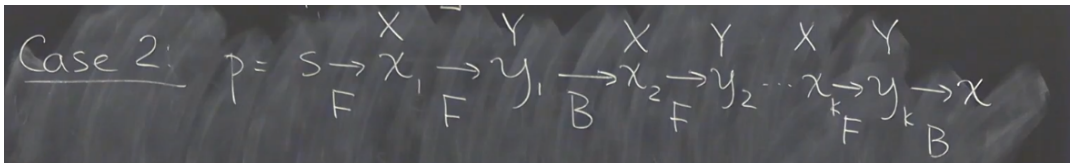
- Why no type 3 edges?
  - For contradiction, assume (x, y) is an edge in F s.t. $x \in X \cap S$ and $y \in Y \cap T$
  - $x \in S \implies \exists s \to x$ path P in G'_f

    Then we have 2 cases,

    1. $P = s, x$, then $f(s, x) = 0$, since we have integral flow with capacity 1, and having a forward edge in G'_f must mean that we have flow of 0 going through that edge.

       Since $y \notin S$, G'_f has BW edge $(y, x) \implies f(x, y) = 1$

       Then we have a contradiction, because flow into $x = 0$, but flow out of $x \geq 1$
    2. We have



       Then $f(x, y_k) = 1$ and $f(x, y) = 1$, but $y \neq y_k$ $[y \notin S, y_k \in S]$. So flow out of $x \geq 2$ but flow into $x \leq 1$, again arriving at a contradiction.
  - Thus, type 3 edges cannot exist.

1.3 Algorithm

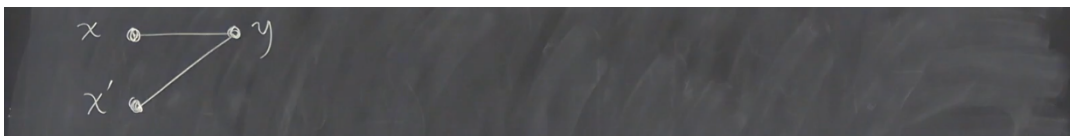
```
1: procedure MIN V.C.(G = ((X, Y), E))
2:     Construct F from G                          ▷ O(m + n)
3:     Find max flow f ∈ F                         ▷ O(mn)
4:     Find min cut (S, T) ∈ F                     ▷ O(m + n)
5:     return (X ∩ T) ∪ (Y ∩ S)                    ▷ O(m)
```

Running time is $O(mn)$

2 Hall's Theorem

2.1 Theorem

- Suppose we have a bipartite graph and the following case happens



We cannot have a perfect matching because only one $x \in X'$ can be matched with $y \in Y$

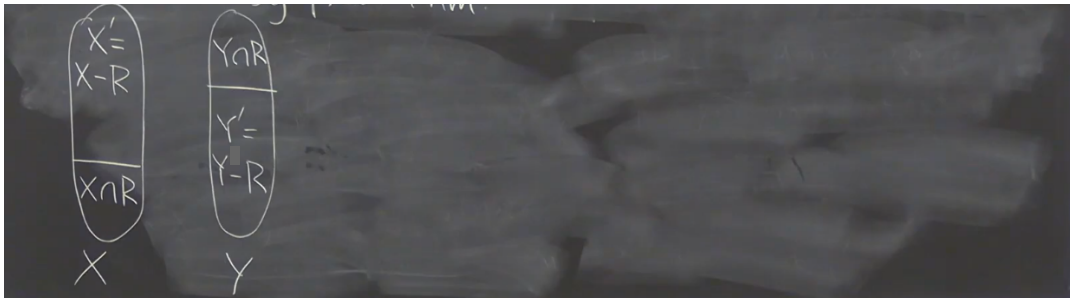
- In general,

  $\exists X' \subseteq X$ s.t. $|N(X')| < |X'|$ where $N(X') = \{y : (x, y) \in E \ \& \ x \in X'\} \iff G$ has no perfect matching ($|M| = |X| = |Y|$)

2.2 Proof

($\Longleftarrow$)

- Let R = min v.c. of G, then we have that $|R|$ = |max matching| by previous theorem.
- In addition, let |max matching| $< \frac{n}{2}$ because we need at least $|X| = |Y|$ to be guaranteed a perfect matching. $|R| < \frac{n}{2}$ $(\star)$
- Consider:



  Will show that $|X'| > |N(X')|$
- R' is a v.c. $\implies$ no $X' \to Y'$ edge in G $\implies$ $N(X') \subseteq Y \cap R$ $\implies$ $|N(X')| \leq |Y \cap R|$

$$\begin{aligned} |N(X')| &\leq |Y \cap R| \\ &= |R| - |X \cap R| \\ &= |R| - (|X| - |X'|) \\ &= |R| - |X| + |X'| \\ &< |X'| \quad (\text{since } |R| < \frac{n}{2}, |X| = \frac{n}{2} \implies |R| - |X| < 0) \end{aligned}$$

($\Longrightarrow$) should be obvious

3 Finding Max Set of Edge-Disjoint Paths

3.1 Problem

- **Input**: $G = (V, E)$ digraph; $s, t \in V$ where $s \neq t$
- **Output**: Max cardinality set of edge disjoint $s \to t$ paths (no edge in common).
- Consider graph

There are 2 edge-disj $s \to t$ paths

3.2 Solution

- Note that we cannot have edges into s or out of t for this problem.

- Algorithm:
  - From G construct $F = (G' = (V, E'), s, t, c)$
    $G' =$ same nodes as G but delete edges into s and out of t
    $c(e) = 1, \forall e \in E'$

3.3 Algorithm

```
1: procedure MaxPaths(G, s, t)
2:     Construct F as shown before
3:     f ← MaxFlow(F)
4:     'Decompose' f into set P of V(f) edge-disjoint simple s → t paths
5:     return P
```

3.4 Theorem

- We have that

  integral flow of value k in $F \iff$ set of k edge-disj simple $s \to t$ paths in G

- Proof:

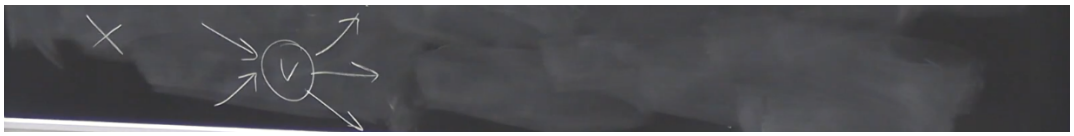
  $\Longleftarrow$
  - Given set P of k edge-disj simple path, define flow f s.t. $V(f) = k$
  - Simply define

$$f(e) = \begin{cases} 1 & e \text{ is on a path in } P \\ 0 & \text{otherwise} \end{cases}$$

  - Claim that f is flow. Need capacity and conservation.
    * capacity is trivial, because of our definition of $f(e)$

* for conservation, consider

Let the edges going in be pigeons and let edges going out be holes. Then by pigeon hole principle, two edges going in must share an edge going out. This is by the fact that we have k edge-disj simple paths.



This is also impossible by pigeon hole principle. Edges going out are pigeons and edges going in are holes.

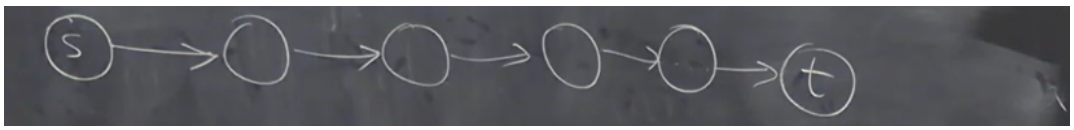
– Now we consider

$$\begin{aligned} V(f) &= \sum_{e \in out(s)} f(e) \\ &= \text{number of edges out of } s \text{ on paths in } P \\ &= |P| = k \end{aligned}$$

$\Longrightarrow$

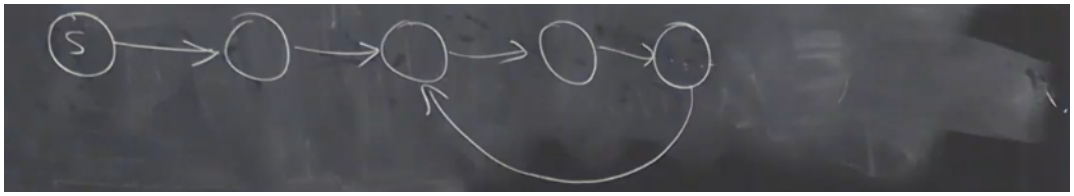
– Given integral flow of value k, find set P of k edge-disj simple $s \to t$ paths s.t. $\forall$ edge e on a path in P, $f(e) = 1$

– Consider the first case:



Simply, remove traffic from all edges on this path and add the path into the set P

– Consider the second case:



We have a cycle, which is useless, so remove all traffic of edges within the cycle.

– Do induction on the number of edges that carry traffic to get set P of decomposed paths. Until we reach 0 traffic for all edges.

3.5 Path Decomposition Algorithm

```
1:  procedure PathDecomp(f)
2:      if (V(f) = 0) then
3:          return ∅
4:      else
5:          u ← s
6:          p ← s
7:          mark all nodes as "unvisited"
8:          while (u is unvisited and u ≠ t) do
9:              mark u as "visited"
10:             v ← any node s.t. f(u, v) = 1
11:             u ← v
12:             p ← p ⊕ v                          ▷ concatenation
13:         if (u = t) then                        ▷ found path from s to t
14:             for (each edge e on p) do          ▷ zero out edges on path
15:                 f(e) ← 0
16:             return PathDecomp(f) ∪ {p}
17:         else                                   ▷ found cycle
18:             for (each edge e on suffix of p from u to u) do   ▷ zero out edges on cycle
19:                 f(e) ← 0
20:             return PathDecomp(f)
```

3.6 Complete Algorithm

```
1:  procedure MaxPaths(G, s, t)
2:      Construct F as shown before                ▷ O(m + n)
3:      f ← MaxFlow(F)                             ▷ O(mn)
4:      P ← PathDecomp(f)                          ▷ O(mn)
5:      return P
```

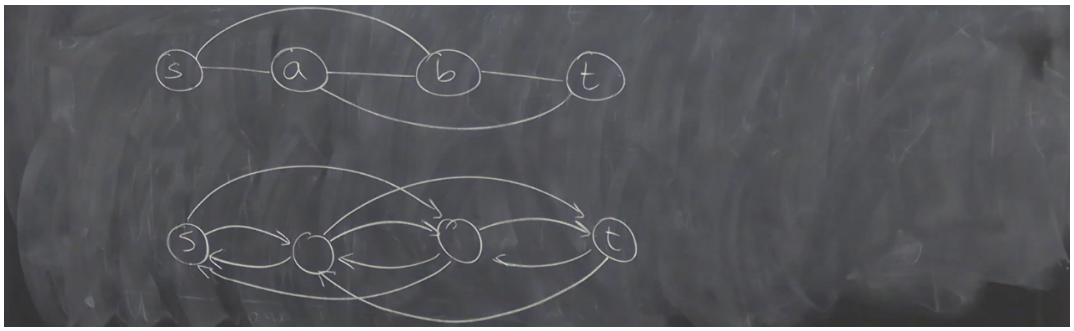
3.7 Correctness

- By line 2, P is a set of $V(f)$ edge-disj $s \to t$ paths
- By line 3, there is no bigger set of edge-disj $s \to t$ paths

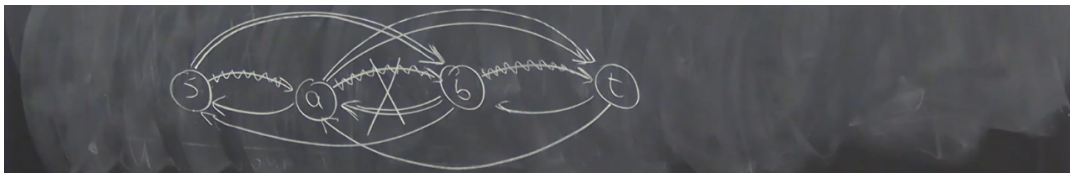
3.8 For undirected graphs

- To construct an undirected graph from a digraph, we typically replace each edge with 2 opposite edges.

  Consider this example, our algorithm might consider illegal paths. //



  Here will not have edge-disjoint paths



  We have the problem where 2 edges actually correspond to 1 edge. To resolve this problem, consider the graph as a flow network, and simply remove the cycles in the flow network (which is useless anyways).