Tuesday, October 18

1. Welcome!
2. HW5 due tonight-ish
3. Looking Ahead
   - next Tues.: finish matchings
   - next Thurs: work day for project
     ↳ bring some resources!
4. Questions?
5. Matchings
6. Outro
   ↳ finish HW!

Today we're still talking about matchings, but we're pivoting from proving structural results to our typical treatment of building a linear program and working with a specialized algorithm. Today we'll focus will be on matchings in bipartite graphs. This is because it has some really tractable algorithms, along with an interesting interpretation of the dual. We'll come back to matchings in general graphs at the very end of our discussion. Let's get going!

Example: Write an integer program that calculates the size of the largest matching in a graph G.

$$\begin{cases} \max \sum x_e \\ \text{st} \sum_{e \sim v} x_e \le 1 \quad \forall v \in V \\ x_e \ge 0,\ x_e \in \mathbb{Z}. \end{cases}$$

vertex $\begin{bmatrix} & \\ & \end{bmatrix}$ edges

$$\begin{cases} \min \sum y_v \\ \text{st } y_u + y_v \ge c_{uv} \quad \forall e = uv \in E \\ y_v \ge 0 \end{cases}$$

vert

edges $\begin{bmatrix} \\ A^T \\ \\ \end{bmatrix} \begin{bmatrix} \\ y \\ \\ \end{bmatrix} \ge \begin{bmatrix} \\ c \\ \\ \end{bmatrix}$

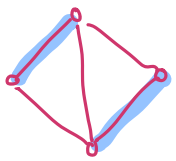
Follow up: Suppose we wanted to find a maximum matching in a weighted graph. How would your program change? Using this change, relax it to a linear program and find the dual in the space above.

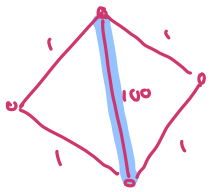
$$\begin{cases} \max \sum c_e x_e \\ \text{st} \sum_{e \sim v} x_e \le 1 \quad \forall v \\ x_e \ge 0,\ x_e \in \mathbb{Z}. \end{cases}$$

edges

vert $\begin{bmatrix} \\ \square \\ \downarrow \\ \begin{cases} 1 & v \in e \\ 0 & v \notin e \end{cases} \end{bmatrix}$ A $\begin{bmatrix} \\ x_e \\ \\ \end{bmatrix} \le \begin{bmatrix} \\ \vdots \\ \\ \end{bmatrix}$

Follow follow up: Draw a small graph, find its maximum matching, then weight it such that the maximum *weighted* matching uses fewer edges but has a larger weight.





Notice! The dual program has a vertex-indexed decision variable. It might make us think that we can give the program some interpretation. For a general graph G, it's not particularly clear, because the dual does not need to be integral.

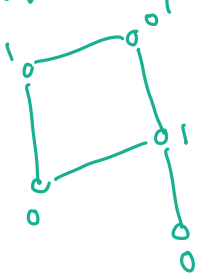
However, if G is bipartite, the dual does give some context about the graph! That's because the incidence matrix for a bipartite graph G is *totally unimodular*. This means all square nonsingular submatrices have determinant ± 1. This is a super big deal, because it means the vertices of the polyhedron are integral!

Example: Rewrite your dual program from the first page in the space below, using the idea that every edge has cost 1 (or, equivalently, a no cost version). Knowing that this dual *must be integral*, interpret the dual.

$$\begin{cases} \min \sum y_v \\ \text{st } y_u + y_v \geq 1 \quad \forall e = \{u,v\} \in E \\ y_v \geq 0, \; y_v \in \mathbb{Z} \end{cases}$$

every edge is incident to a picked vertex!

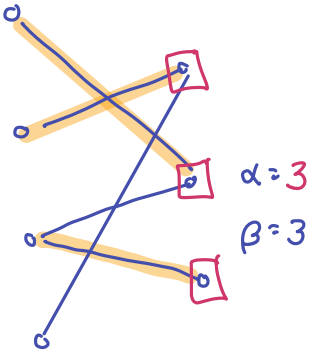
totally unimodular.

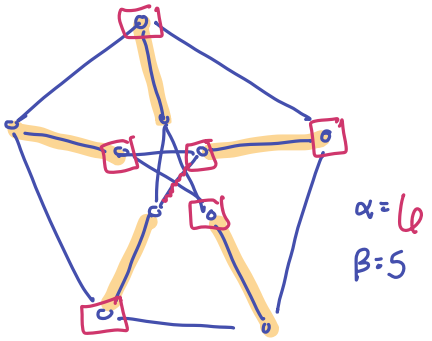


So certainly some result is kicking around here! Let's scaffold it out with some definitions.

Definition: (vertex cover) A vertex cover is a set of vertices $S \subseteq V$ such that every edge of G is incident to some vertex in S. The size of the smallest vertex cover of G is denoted $\alpha(G)$.

Example: Draw (a) a bipartite graph on 6 or more vertices and (b) the Petersen graph. Find the maximum matching and minimum vertex cover of these graphs.





Follow up: Suppose we let $\beta(G)$ be the size of the largest matching in any graph. Why does it make sense that $\alpha(G) \geq \beta(G)$ (a) from a graph theoretic perspective and (b) from a linear programming perspective?

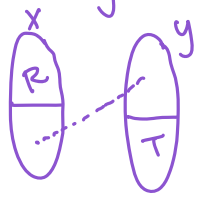
a) for every edge in the matching, we need to pick a vertex for the cover. So $\alpha \geq \beta$.

b) dual is an upper bound for primal. a vertex cover is a feas. sol. for the dual, (but may not be optimal)

Theorem (König)**.** *Let G be a bipartite graph. Then $\alpha(G) = \beta(G)$.*

Proof: (1 of 2) Construct the integer program, realize that relaxing it to the linear program does not change the solutions (thanks totally unimodular matrices), and interpret the dual as a vertex cover.

Proof: (2 of 2) Note, we know $\alpha(G) \geq \beta(G)$, so it suffices to show if G b/p, then $\alpha(G) \leq \beta(G)$. Let S be a min vertex cover, s.t. $|S| = \alpha$. We will construct a matching of that size. Let $R = S \cap X$, $T = Y \cap S$.



note, no edges b/t $X \setminus R$ and $Y \setminus T$, else not covered! Consider 2 subgraphs

Cl: $\exists$ matchings that saturate R and T.

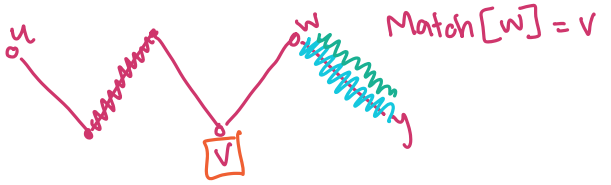
↳ Pf: Hall's Thm and $\lightning$.

So our matching is size $|R| + |T| = |S|$.

So let's talk algorithms: how do we find a maximum matching in a bipartite graph? One option is the Hungarian Algorithm, which relies on the idea that a maximum matching has no augmenting paths., proven by us in Homework 5. This will need a bit of a helper method, Augment, along with a dash of BFS.

Algorithm 1 Augment(y)

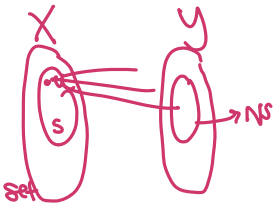
1: **repeat**
2: $w \leftarrow PrevPt[y]$
3: $Match[y] \leftarrow w$
4: $v \leftarrow Match[w]$
5: $Match[w] \leftarrow y$
6: $y \leftarrow v$
7: **until** $y = \emptyset$

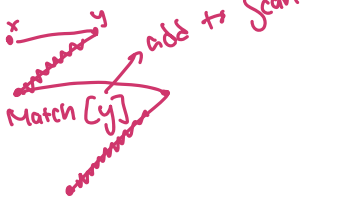


Notes: this augments an aug. path from end to start.

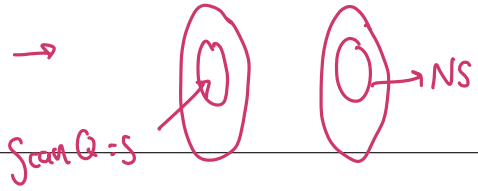
Algorithm 2 MaxMatching($G = (X \cup Y, E)$,n)

for $i \leftarrow 1$ **to** n **do**
  $Match[i] \leftarrow 0$ ← nothing in matching so far.
end for
for all $u \in X$ **do**
  $ScanQ[1] \leftarrow u$
  $QSize[\,] \leftarrow 1$
  for $i \leftarrow 1$ **to** n **do**
    $PrevPt[i] \leftarrow 0$ ← set predecessor to zero for all vert.
  end for
  $k \leftarrow 1$
  repeat
    $x \leftarrow ScanQ[k]$
    for all $y \rightarrow x$ **do**
      if $y \notin NS$ **then**
        Add y to NS : put it in reachable set
        $PrevPt[y] \leftarrow x$
        if y is unsaturated **then** yay! aug. path!
          Augment(y)
          go to (1)
        end if
        Add $Match[y]$ to $ScanQ$. ! ← ∈ x
        $QSize \leftarrow QSize + 1$.
      end if
    end for
    $k \leftarrow k + 1$
  until $k > QSize$
  Delete $ScanQ$ and NS from the graph. →
  (1)
end for

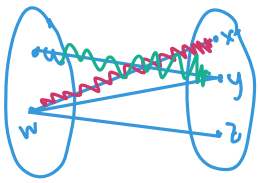




forces the path from u to alternate.



Example: Run one step of the algorithm, starting at u, on the graph below.



ScanQ: u, w

NS: y, x

PrvsPt[y] = u

PrvsPt[x] = w

Match[x] = w

Match[w] = x (was y)

Match[y] = u

match[u] = y