18.200 lecture notes March 18, 2022

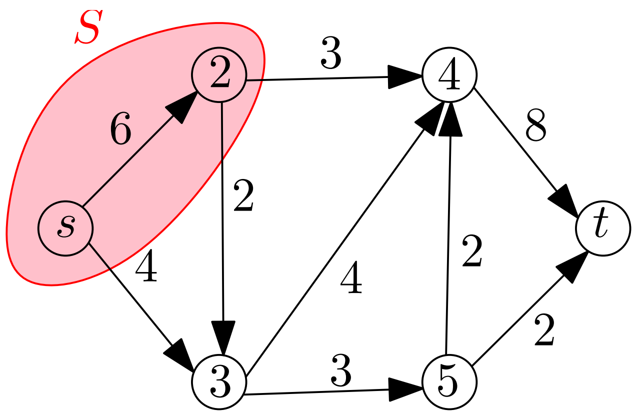
Network flows and matching

Lecturer: Peter Shor

1 The Maximum Flow Problem

Imagine (if you will) that the sole purpose of the internet is to allow you to stream Netflix. Today we're going to tackle the following problem: How can we ensure that you can stream at the maximum possible quality? Let's formalize this problem as follows. A *network* consists of a set V of vertices (plural of *vertex*...), and set A of arcs (or directed edges) between vertices: an arc from i to j is represented by the ordered pair $(i, j) \in A$. Each arc $(i, j) \in A$ has a *capacity* denoted by $u(i, j)$, representing the maximum rate at which we can send "things" along this arc. You can think of the vertices as representing routers, and the capacity of an arc represents the strength of the connection (what rate router i can transmit to router j).

Here is an example network:



In this network we have two special vertices: a *source* vertex s and a *sink* vertex t. In this example, Netflix would be the source and your home computer would be the sink. Our goal is to send the maximum amount of flow from s to t; What is a flow exactly? Mathematically it is a function $f : A \to \Re^+$ that assigns nonnegative values to each arc, and we require that it satisfies two important properties: The first property is *flow conservation.* Consider some vertex i, different from the source or the sink. Then the amount of flow going into vertex i should be the same as the amount of flow going out; no flow appears or disappears at this vertex. The second property is *capacity constraints.* We require that for each arc (i, j) the amount of flow does not exceed its capacity – i.e. $f(i, j) \leq u(i, j)$.

Among all possible flows that meet these constraints, which one sends data to you at the maximum possible rate? In the above example, there is a flow of value 9 (you should be able to find such a flow pretty easily). But there is no flow of any larger value. Later we will see how we can formally prove this.

In fact "flow" could be many other things: imagine sending water along pipes, with the capacity representing the size of the pipe; or traffic, with the capacity being the number of lanes of a road. Network flows appear in many many settings. In fact, one of the original applications of flow was in routing troops in East Germany during the Cold War, and it came up as a natural optimization problem in this context when the US wanted to understand how quickly the Russians could mobilize their troops and also what railroad lines they should destroy to curtail the flow of troops the most effectively.

2 Max Flow as a Linear Program

Our first goal is to express the maximum flow problem as a linear program. We will then be able to leverage the tools we've developed (e.g. duality) to gain further insights about the maximum flow problem. Our decision variables will be

$$\begin{aligned} x_{ij} &= \text{amount of flow on arc } (i,j) \in A \\ z &= \text{total amount of flow sent from } s \text{ to } t. \end{aligned}$$

The trickiest constraint to deal with is flow conservation. Consider any vertex i different from s and t; since flow out is equal to flow in, we have

$$\sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji};$$

note the order of the indices, it matters! What about if $i = s$? Then the amount of flow going out should be larger than the amount coming in, by an amount z. (You might expect that there should be no flow at all coming in, and you would be right; but we don't need to exclude the possibility at the moment, and it makes the equations easier in fact.) So we have

$$\sum_{j:(s,j)\in A} x_{sj} = \sum_{j:(j,s)\in A} x_{js} + z.$$

Similarly,

$$\sum_{j:(t,j)\in A} x_{tj} = \sum_{j:(j,t)\in A} x_{jt} - z;$$

note the change in sign. We can put all three of these equations together as one, using some notation:

$$\sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} \; + \; (\mathbf{1}_{i=s} - \mathbf{1}_{i=t})z.$$

Here, $\mathbf{1}_E$ is just the indicator of the statement E; it is 1 if E is true, and 0 otherwise.

The other important constraint we must remember is simply that the flow on an arc must not exceed the capacity. Putting this together, we have the following formulation of the maximum flow problem:

$$\begin{aligned} \max \quad & z \\ \text{subject to} \quad & \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} - (\mathbf{1}_{i=s} - \mathbf{1}_{i=t})z = 0 \qquad \forall i \in V \qquad (1) \\ & x_{ij} \le u(i,j) \qquad \forall (i,j) \in A \\ & z, x \ge 0. \end{aligned}$$

This already tells us that we can solve the maximum flow problem using algorithms for solving general linear programs. There are, however, much faster algorithms for solving this problem.

3 s-t Cuts

Now returning to the example in the Figure, we asserted that there is a flow of value 9 and that there is no flow of any larger value. Intuitively, we can see that this is the case by considering the set S shown in red on the figure. Since $s \in S$ and $t \notin S$, all flow from s to t needs to cross the boundary of S. In the example, the total amount of outgoing capacity crossing this boundary is 9; and so there cannot be a flow of larger value.

We can formalize this claim that the value of any flow is at most the capacity of any cut.

Lemma 1. *Given a flow x of value z and an $s-t$-cut S, we have $z \le \text{cap}(S)$.*

Proof. Given x and S, we obtain by summing (1) over $i \in S$:

$$z = \sum_{(i,j)\in A: i\in S, j\notin S} x_{ij} - \sum_{(j,i)\in A: j\notin S, i\in S} x_{ji}. \qquad (2)$$

Using the fact that $x_{ij} \le u(i,j)$ for the arcs in the first summation, and $x_{ji} \ge 0$ for the arcs in the second summation, we obtain:

$$z \le \sum_{(i,j)\in A: i\in S, j\notin S} u(i,j),$$

showing the claim as the last term is precisely $\text{cap}(S)$. □

Let's formalize this notion a bit.

Definition 1. A *s-t-cut* is a subset $S \subset V$ such that $s \in S$, $t \notin S$. The *capacity* of an s-t-cut S is

$$\text{cap}(S) = \sum_{(i,j)\in A: i\in S, j\notin S} u(i,j).$$

Note that we only count *outgoing* arcs towards the capacity of a cut; flow must traverse arcs in the forward direction, so incoming arcs are not useful in "escaping" the cut. The total amount of flow from s to t can't exceed the capacity of any s-t-cut. This is rather intuitive, but will be derived formally in the next section.

We use some shorthand and use "min cut" to refer to the minimum capacity of any s-t-cut. So we have

$$\text{max flow} \quad \le \quad \text{min cut}.$$

But more is true!

Theorem 1. *max flow* $=$ *min cut.*

We will show two ways to prove this theorem. One is to express the problem of finding a maximum flow as a linear program, and use strong LP duality. The other one is to look at properties of maximum flows and show that we can derive from any maximum flow a corresponding minimum $(s-t)$ cut giving the same value.

4 The Dual

In this section, we provide a proof of the maximum flow minimum cut theorem given in Theorem 1 using linear programming duality.

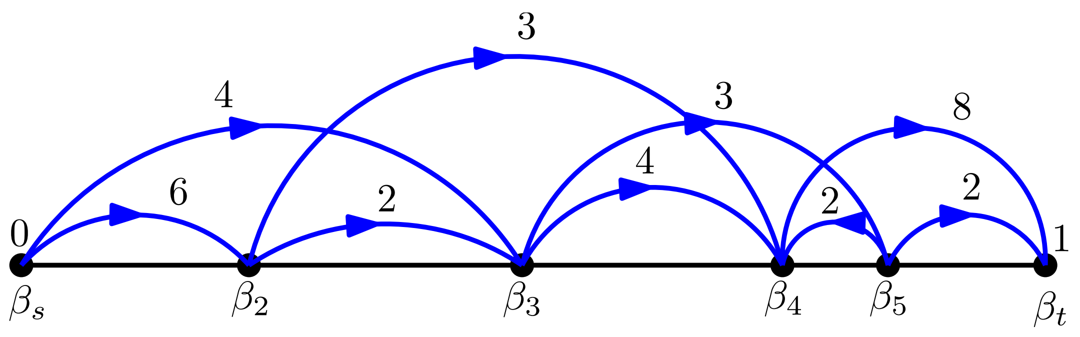
The first step step is to write the dual. One way to do this is to write the LP in matrix form, replacing equality constraints by pairs of inequalities (since the constraint $a^Tx = b$ is the same as the pair of constraints $a^Tx \le b$, $a^Tx \ge b$). This is certainly doable, but a little painful. If you work with linear programs enough, you get quite good at taking the dual; there are some discussion about shortcuts in your linear programming notes. If you consult those, you should be able to determine that the following is indeed the dual:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} u(i,j)\alpha_{ij} \\ \text{subject to} \quad & \beta_t - \beta_s \ge 1 \\ \alpha_{ij} + \beta_i - \beta_j &\ge 0 \qquad \forall (i,j) \in A \\ \alpha_{ij} &\ge 0 \qquad \forall (i,j) \in A \\ \beta_i &\in \mathbb{R} \qquad \forall i \in V. \end{aligned}$$

So far, we see no sign of cuts; somehow we need to massage our dual if we're going to get our max-flow min-cut theorem.

First, notice that for a fixed $(i,j) \in A$, the only constraints on α_{ij} are that $\alpha_{ij} \ge \beta_j - \beta_i$, and $\alpha_{ij} \ge 0$. Let's use the convenient notation $(Z)^+ := \max(Z, 0)$. So we have that $\alpha_{ij} \ge (\beta_j - \beta_i)^+$. Next, observe that just by shifting all the β_i's by the same amount, we can obtain an equivalent solution with $\beta_s = 0$, so let's assume this from now on.

Each β_i is associated with a vertex $i \in V$. So we can imagine drawing the graph, in such a way that vertex i is placed at position β_i:



Now we can interpret the objective nicely in this picture. Any arc that is going from right to left contributes nothing to the dual objective. An arc (i,j) going from left to right contributes an amount equal to the product of its capacity $u(i,j)$ and its length $\beta_j - \beta_i$. Thus, independently of whether the arc (i,j) goes to the right or to the left, it contributes $u(i,j)(\beta_j - \beta_i)^+$ to the dual objective function.

Observe that any $s-t$-cut S corresponds to a solution to the dual in which β_i is 0 for $i \in S$ and 1 for $i \notin S$. But the dual contains many solutions β that do not correspond to cuts. Consider any optimum solution β^* in the dual. The β_i^* values might not be all 0 or 1, but we show below that we can extract from it an $s-t$-cut that has the same dual objective function value.

Given an optimum dual solution β^*, define, for each $q \in [0,1)$, the s-t-cut

$$S(q) := \{i \in V \mid \beta_i^* \le q\}.$$

(This is certainly an s-t-cut, since $\beta_s^* = 0$ and $\beta_t^* \ge 1$.) Suppose we generate $q \in [0,1)$ uniformly at random. We have only seen discrete probability, but this is analogous (one could imagine that we finely discretize the interval $[0,1)$ and choose any of these values uniformly at random). The key is to bound the probability that an arc (i,j) belongs to the cut generated by $S(q)$. What is this probability?

Lemma 2. *The probability that the arc (i,j) is cut is at most α_{ij}^*.*

Proof. We will prove this by a case analysis. If $\beta_i^* > \beta_j^*$ then any cut $S(q)$ that contains i also contains j and the arc (i,j) is never cut. If instead $\beta_j^* > \beta_i^*$ then the probability that the arc (i,j) belongs to the cut is exactly

$$(\min(\beta_j^*, 1) - \max(\beta_i^*, 0))^+$$

Now we have $\min(\beta_j^*, 1) - \max(\beta_i^*, 0) \le \beta_j^* - \beta_i^*$. And so the probability that (i,j) belongs to the cut is bounded by $(\beta_j^* - \beta_i^*)^+$ which in turn is at most $\alpha_{i,j}^*$ as we wanted. □

Now by linearity of expectation we have

$$\mathbb{E}(cap(S_q)) \le \sum_{(i,j)\in A} u(i,j)\alpha_{ij}^*. \tag{3}$$

What we're saying is that if we are given any feasible solution to the dual with some objective value C we can round it (randomly) to an s-t-cut in such a way that the expected capacity of the cut is at most C. So appealing to linear programming duality we know that the optimum value of the dual is equal to the maximum flow, and what we have just shown is that we can produce a cut whose capacity is at most the maximum flow. Recall that in Lemma 1 we showed that

$$\text{max flow} \quad \le \quad \text{min cut}$$

and what we just showed via a randomized rounding is that

$$\text{max flow} \quad = \quad \text{optimum dual value} \quad \ge \quad \text{min cut}$$

This completes the proof of Theorem 1. But this also seems a bit strange. Why couldn't the randomized rounding produce an s-t-cut whose value is strictly less than the optimum dual value and thus strictly less than the maximum flow too? Well, it's because the only places where we have inequalities must actually be equalities. We cannot have $\alpha_{ij} > (\beta_j - \beta_i)^+$ on any arc (i,j) where $u(i,j) > 0$, otherwise we could instead pick $\alpha_{ij} = (\beta_j - \beta_i)^+$ while maintaining feasibility of the dual and yet strictly decreasing the objective value.

5 Augmenting path algorithm

Since the maximum flow problem can be formulated as a linear program, we can use the simplex algorithm to solve it. However, one can design more efficient algorithms that use the structure of the linear program. We present here the main concept here to design such algorithms.

A natural algorithm for the maximum flow problem is to start with the zero flow (no flow on any arc) and to repeatedly *push* more flow from the source to the sink until we are unable to push more flow. The notion of *pushing* flow needs, of course, to be formalized.

Suppose we have a flow x. If there exists an arc (s, v_1) with $x_{sv_1} < u(s, v_1)$ then we can increase the flow along this arc say by $\epsilon > 0$ where $\epsilon \leq u(s, v_1) - x_{sv_1}$, thereby increasing the net flow leaving s by ϵ. However, flow conservation at v_1 is not satisfied any more and to repair it, we could increase the flow by ϵ on some arc $(v_1, v_2) \in A$ with $x_{v_1 v_2} < u(v_1, v_2)$ provided ϵ is small enough. Another option to reestablish flow conservation at v_1 would be to *decrease* the flow on an arc $(v_2, v_1) \in A$ with $x_{v_2 v_1} > 0$ by ϵ. We have now flow conservation back at v_1, but we have again destroyed flow conservation at v_2. We can thus proceed similarly and find either an arc $(v_2, v_3) \in A$ on which we can increase flow by ϵ or an arc $(v_3, v_2) \in P$ on which we can decrease flow by ϵ. This operation of pushing flow would be successful if we can eventually reach vertex t, as we do not need to maintain flow conservation at the sink.

To formalize this operation of flow augmentation, it is convenient to define an auxiliary directed graph G_x, called the *residual network*, and which depends on the current flow x. The graph G_x has the same vertex set as G, and its edge set is

$$A_x = \{(i, j) : (i, j) \in A, x_{ij} < u_{ij}\} \cup \{(i, j) : (j, i) \in A, x_{ji} > 0\}.$$

The edges in the first set in this definition are sometimes called the forward arcs and those in the second set the backard arcs. This means that for any $(i, j) \in A_x$, we can either increase the flow on (i, j) or decrease the flow on (j, i). In both cases, the net flow out of i increases while the net flow out of j correspondingly decreases. For each arc (i, j) in A_x, we define a residual capacity $u_x(i, j)$ by

$$u_x(i, j) = \begin{cases} u(i, j) - x_{ij} & (i, j) \in A : x_{ij} < u(i, j) \\ x_{ji} & (j, i) \in A : x_{ji} > 0. \end{cases}$$

An *augmenting path* P with respect to a flow x is defined as a directed path from s to t in the residual graph G_x. Given an augmenting path P we can modify the flow along P, by increasing the flow by $\epsilon(P)$ on forward edges and decreasing it by $\epsilon(P)$ on backward where $\epsilon(P) = \min_{(i,j) \in P} u_x(i, j)$. This choice of ϵ corresponds to the largest value such that the resulting flow satisfies the capacity and nonnegativity constraints.

The *augmenting path* algorithm for the maximum flow problem is very simple to state. Start with the zero flow $x = 0$. While there exists an augmenting path P in the residual graph G_x corresponding to the current flow x, push flow along x. In the case that several augmenting paths exist, we have not specified which one to select. However, independently of which augmenting path is selected, we claim that this algorithm terminates and outputs a maximum flow, and this is stated in the following theorem. We should emphasize, however, that this assumes that the capacities are all integers (or rational numbers). In the case of potentially irrational capacities, an unfortunate choice of augmenting path may lead to the algorithm never terminating, and the flow value converging in the limit to a non-optimal value. When a flow has integer values for every arc, we say that the flow is *integral*.

Theorem 2. *Given a maximum flow problem with integer capacities, the augmenting path algorithm terminates with a flow x^* that is both maximal and integral. Furthermore, there exists a cut S separating s and t whose capacity equals the value of the maximum flow x^*.*

Proof. When the capacities are all integral, we first claim that the flow x will always be integral during the execution of the algorithm. This is shown by induction. Indeed, the algorithm starts with an integral flow ($x_a = 0$ for every $a \in A$), and so the base case is satisfied. Now, assuming that the current flow x is integral, we observe that all residual capacities are integral, and therefore $\epsilon(P)$ is also an integer. Therefore, the flow obtained after pushing $\epsilon(P)$ units along P remains integral.

This implies that the algorithm terminates. Indeed, at every iteration $\epsilon(P)$ is a non-zero integer, and therefore is at least 1. This means that the net flow out of s increases by at least one unit at every iteration, and since the flow value of any flow is bounded by the capacity of any cut, say the cut induced by $\{s\}$, the number of iterations before termination is finite. Let x^* be the integral flow that the algorithm returns.

To show that this flow is a maximum flow, we exhibit a cut separating s from t whose capacity equals the flow value. For this purpose, consider the residual graph G_{x^*} corresponding to this final flow x^*. Define S by

$$S := \{v \in V : \text{there exists a directed path from } s \text{ to } v \text{ in } G_{x^*}\}.$$

Obviously $s \in S$ and, since there are no augmenting paths in G_{x^*} (because of termination of the algorithm), t is not reachable from s and thus $t \notin S$. Thus S induces a cut separating s from t. By definition of S, there are no arcs between S and $V \setminus S$ in the residual graph G_{A^*}. This means that $x^*_{ij} = u(i,j)$ for $(i,j) \in A$, $i \in S$, $j \notin S$, and $x^*_{ji} = 0$ for $(j,i) \in A$, $j \notin S$, $i \in S$. This observation together with (2) imply that

$$z^* = \sum_{(i,j)\in A: i\in S, j\notin S} x^*_{ij} - \sum_{(j,i)\in A: j\notin S, i\in S} x^*_{ji} = \sum_{(i,j)\in A: i\in S, j\notin S} u_{ij} = \text{cap}(S).$$

This means that both x^* is a maximum flow and S gives a mimum $s - t$ cut. □

6 Bipartite Matching

In the previous sections, we have seen that the maximum flow from s to t is equal to the capacity of the minimum cut separating s from t, and that the maximum flow can be assumed to be integral if the capacities are integral. These results have important implications and applications in discrete mathematics and graph theory. Here, we'll see what it means for the *maximum matching problem in bipartite graphs.*

While we were considering directed graphs (or networks) when discussing maximum flow problem, we now consider undirected graphs. A graph $G = (V, E)$ with vertex set V and edge set E is said to be *bipartite* if V can be partitioned into $V = P \cup Q$ such that all edges have precisely one endpoint in P and one in Q: $E \subseteq P \times Q$.

Definition 2. Given a graph $G = (V, E)$, a matching is a set $M \subseteq E$ of edges where none of the edges share an endpoint. The size of a matching of a matching is the number of its edges, and the maximum matching problem asks to find a matching of maximum size.

For general (i.e. not necessarily bipartite) graphs, this is a very interesting problem, both combinatorially and algorithmically, which can be solved efficiently. This, however, is quite involved and will not be covered here. However, for the case of bipartite graphs, the problem easily can be formulated as a maximum flow problem, thanks to the integrality of maximum flows.

Indeed, given a bipartite graph $G = (V, E)$ with bipartition $V = P \cup Q$, consider the following network H. The vertex set of H consists of V and two additional vertices, the source s and the sink t. The arcs of H are of three types: we have an arc (s, p) for every $p \in P$, an arc (q, t) for every $q \in Q$, and all edges in E are directed from P to Q. Let A be the resulting arcs set of H. To define the network, we also need to set the capacities on all the arcs. We let $u(s, p) = 1$ for all $p \in P$, $u(q, t) = 1$ for all $q \in Q$, and all the other arcs a get an integral capacity at least 1, for example $u(a) = 1$ although we'll see later that setting them arbitrarily large ($u(a) = +\infty$) could be advantageous.

With this construction of H, notice that we have a *bijection* between *integral flows* x of value k and matchings M of size k. Indeed, if we have an integral $s - t$ flow x, let $M = \{(p, q) \in P \times Q : x_{pq} = 1\}$. From flow conservation and integrality, we see that M is indeed a matching (i.e. has at most one edge incident to every vertex in P and in Q), and its size is precisely the flow value of x. Conversely, given a matching M of size k, one can construct a flow x of value k by simply setting $x_{pq} := 1$ if $(p, q) \in M$ and 0 otherwise, and $x_{sp} := \sum_q x_{pq}$ for all $p \in P$, and $x_{qt} := \sum_p x_{pq}$ for all $q \in Q$. The setting for the arcs incident to s and t insures that flow conservation is satisfied. Notice that this bijection between integral flows in H and matchings in the (bipartite) graph G holds, irrespective of how the integral capacities were set for the arcs from P to Q (provided they are at least 1).

Given this bijection between integral flows and matchings, the maximum matching problem problem is thus equivalent to a maximum flow problem, thanks to the integrality property of maximum flows. We can also interpret the maxflow-mincut theorem in several ways. We first need the following definition.

Definition 3. Given a graph $G = (V, E)$, a vertex cover C is a subset $C \subseteq V$ such that every edge $e \in E$ is incident to at least one vertex of C.

Given a matching M and a vertex cover C, we must have $|M| \leq |C|$ since a vertex cover must contain at least one of the endpoints of each edge in M, and these edges of M do not share any endpoint, given that M is a matching. This means that the size of a maximum matching is at most the size of the smallest vertex cover. This is weak duality. For general graphs, however, we may not have equality; in the complete graph on three vertices, K_3, any matching contains at most one edge while any vertex cover contains at least two vertices. In bipartite graphs, however, strong duality holds and this follows from the maxflow-mincut theorem.

Theorem 3 (König's theorem)**.** *In any bipartite graph $G = (V, E)$ with bipartition $V = P \cup Q$, the maximum size of any matching M equals the minimum size of a vertex cover C.*

Proof. We formulate the maximum matching problem as a maximum flow problem as described above, and for convenience, we set the capacities of the arcs between P Q to be $+\infty$ (rather than 1). We know that the size of the maximum matching equals the capacity of the minimum cut separating s and t. Consider any $s - t$ cut S of minimum capacity; thus $s \in S$ and $S \subseteq \{s\} \cup P \cup Q$. If there is an arc from $S \cap P$ to $Q \setminus S$, then the (infinite) capacity of this arc contributes to the capacity of the cut, and therefore is definitely not a minimum cut. This means that we can assume that our graph G has no edge between $S \cap P$ and $Q \setminus S$. In other words, if we set $C = (P \setminus S) \cup (Q \cap S)$, we obtain that C is a vertex cover. And the size of this vertex cover is precisely that the capacity of the cut defined by S, since S cuts $|P \setminus S|$ arcs incident to the source and $|Q \cap S|$ arcs incident to the sink. This completes the proof of the theorem since we have obtained a vertex cover C of the same cardinality as a matching M, and therefore they both must be optimum given weak duality mentioned previously. $\square$

We can also interpret the maxflow mincut theorem in another way to get a necessary and sufficient condition for the existence of a matching (in a bipartite graph) which *covers* every vertex in P; this is known as Hall's theorem.

Theorem 4 (Hall's theorem)**.** *Given a bipartite graph $G = (V, E)$ with bipartition $V = P \cup Q$, there exists a matching M covering every vertex in P if and only if for every $R \subseteq P$, we have $|\Gamma(R)| \geq |R|$ where*

$$\Gamma(R) := \{q \in Q : \exists p \in R, (p, q) \in E\}$$

if the neighborhood of R.

Proof. The "only if" part should be clear, since the existence of a matching covering P means that the neighborhood of R contains at least $\{q \in Q : (p, q) \in M \text{ with } p \in R\}$.

For the if part, consider any $s - t$ cut S of finite capacity in the network associated with this matching instance. Similarly to the proof of König's theorem, we do not have any edge between $S \cap P$ and $Q \setminus S$, implying that $\Gamma(S \cap P) \subseteq S \cap Q$. The condition that $|\Gamma(R)| \geq |R|$ for any $R \subseteq P$ implies that $|S \cap P| \leq |S \cap Q|$. Therefore the capacity of this cut satisfies:

$$cap(S) = |P \setminus S| + |S \cap Q| = |P| - |S \cap P| + |S \cap Q| \geq |P|,$$

and since there is a cut of capacity $|P|$ (namely the cut separating s from the rest), the minimum $s - t$ cut must have capacity precisely $|P|$. Thus there is a matching covering every vertex of R. $\square$