CME 305: Discrete Mathematics and Algorithms

1 Network Flow

A **network** N is a set containing:

- a directed graph $G(V, E)$;
- a vertex $s \in V$ which has only outgoing edges, we call s the source node;
- a vertex $t \in V$ which has only incoming edges, we call t the sink node;
- a positive capacity function $c : E \mapsto I\!R^+$.

A **flow** f on a network N is a function $f : E \mapsto I\!R^+$. Flow f is a **feasible flow** if it satisfies the following two conditions:

1. **Edge capacity limit:**
$$\forall\ e \in E,\ 0 \leq f(e) \leq c(e)$$
2. **Conservation of flow:**
$$\forall\ v \in V \setminus \{s, t\},\ \sum_{e\ leaving\ v} f(e)\ =\ \sum_{e\ entering\ v} f(e).$$

The **value** of a flow f is defined as

$$v(f)\ \equiv\ \sum_{e\ leaving\ s} f(e).$$

Since s and t are the only nodes that do not conserve flow, the value of f can be equivalently stated as the amount of flow entering t:

Proposition 1 *For any feasible flow,*

$$v(f)\ =\ \sum_{e\ leaving\ s} f(e)\ =\ \sum_{e\ entering\ t} f(e)$$

Proof: This follows directly from conservation of flow.

$$\begin{aligned} v(f)\ &=\ \sum_{e\ leaving\ s} f(e) \\ &=\ \sum_{e\ leaving\ s} f(e) - \sum_{v \in V/\{s,t\}} \left[\sum_{e\ entering\ v} f(e) - \sum_{e\ leaving\ v} f(e) \right] \qquad (1) \\ &=\ \sum_{e\ entering\ t} f(e), \end{aligned}$$

where the last line is due to the fact that each edge e appears twice in (1) - once as a leaving edge (with positive sign) and once as an entering edge (with negative sign) - except those edges entering t which appear exactly once and with positive sign. ■

An **s-t cut** (A, B) is a partition of V into subsets A and B such that $s \in A$ and $t \in B$. We define the **cut value**, $c(A, B)$, to be the sum of capacities of all the edges going from set A to set B.

$$c(A,B) = \sum_{\substack{e \text{ leaving } A, \\ \text{entering } B}} c(e)$$

Remark 1: Using a proof similar to Proposition 1, it may be shown that for any cut (A, B):

$$v(f) = \sum_{\substack{e \text{ leaving } A, \\ \text{entering } B}} f(e) \; - \sum_{\substack{e \text{ leaving } B, \\ \text{entering } A}} f(e) \; .$$

Remark 2: For any feasible flow f and cut (A, B) we note:

$$v(f) = \sum_{\substack{e \text{ leaving } A, \\ \text{entering } B}} f(e) \; - \sum_{\substack{e \text{ leaving } B, \\ \text{entering } A}} f(e) \; \leq \sum_{\substack{e \text{ leaving } A, \\ \text{entering } B}} f(e) \leq \sum_{\substack{e \text{ leaving } A, \\ \text{entering } B}} c(e) = c(A,B)$$

Therefore any s-t cut value is an upper bound on $v(f)$. Given a network N, the **max-flow problem** is to find a feasible flow with the maximum possible value. Remark 2 implies that

$$\max_{f \; feasible} v(f) \leq \min_{(A,B)} c(A,B) \, .$$

We will show that equality is in fact attained by the max-flow and min-cut values.

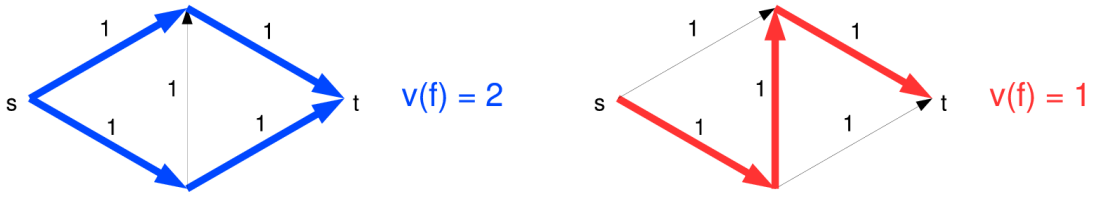
1.1 Ford-Fulkerson Algorithm

In this section we develop the Ford-Fulkerson (FF) algorithm for finding the max-flow in a network. Ford-Fulkerson may be seen as a natural extension of the following simple, but ineffective, greedy algorithm.

Algorithm 1 Greedy Max-Flow Algorithm (Suboptimal)

Initialize $f(e) = 0$ for all $e \in E$.
repeat
  Find path P between s and t such that $min_{e \in P} (c(e) - f(e)) > 0$, we call such a path unsaturated.
  Let $df = min_{e \in P} (c(e) - f(e))$; $f(e) = f(e) + df$, $e \in P$.
until No more unsaturated s-t paths

This greedy algorithm does not find the max-flow in general graphs. A simple counterexample can be seen in Figure 1.



(a) Optimal flow (b) Suboptimal maximal flow

Figure 1: Two potential outcomes of the greedy algorithm. *a*) The optimal flow is achieved. *b*) No more flow can be pushed greedily through the network.

In Figure 1(b), the greedy algorithm has made a bad choice for the first unit of flow to push through. There are no remaining unsaturated s-t paths in the network, but the maximum flow has not been achieved. We modify the algorithm such that we can revise the paths later in the run of the algorithm. This is the rough idea of the Ford-Fulkerson algorithm.

In order to describe the FF algorithm, we first define a residual network of network N with respect to flow f. The **residual network** $R(N, f)$ is a network with vertex set V and with edge set E_r constructed as follows:

For every $e \in E$:

- if $f(e) < c(e)$ place an edge with capacity $c'(e) = c(e) - f(e)$ in the same direction as e.
- if $f(e) > 0$ place an edge with capacity $c'(e) = f(e)$ in the *opposite* direction of e.

The advantage of the residual network $R(N, f)$ is that any path P from s to t in $R(N, f)$ gives a path along which we can increase the flow, including ones that reverse previously assigned flow. Building the residual network and augmenting along an s-t path forms the core of Ford-Fulkerson algorithm.

Algorithm 2 Ford-Fulkerson, 1956

Initialize $f(e) = 0$ for all $e \in E$.
while there is a path P from s to t in $R(N, f)$ **do**
  send a flow of value $df = \min_{e \in P} c'(e)$ in R along P.
  update f in (N, f): set $f(e) = f(e) + df$ $\forall e \in P$
  rebuild the residual network $R(N, f)$.
end while
Output f^*.

Theorem 1 *If Ford-Fulkerson terminates, it outputs a maximum flow.*

Proof: Suppose the algorithm terminates at step t, this means that there is no path from s to t in $R(N, f^*)$; s and t are disconnected. Let S be the set of nodes *reachable* from s in $R(N, f^*)$, i.e., $v \in S$ iff there exists a path from s to v; let $T = V \setminus S$. We claim that $v(f^*) = c(S, T)$. Before proving this claim, we recall that for every feasible flow f and every cut (A, B), $v(f) \leq c(A, B)$ (Remark 2). Thus $v(f^*) = c(S, T)$ implies that f^* is the max-flow and (S, T) is the minimum cut.

In order to prove $v(f^*) = c(S, T)$, first consider any edge e from S to T in the original network N. Edge e must not exist in $R(N, f)$, or else its endpoint in T would be reachable from s, contradicting the definition of T. Thus it must be the case that $f(e) = c(e)$ for any such e. Now we consider e' from T to S in the original network N. If $f(e') > 0$ then there will be an edge in the opposite direction of e' in $R(N, f)$ i.e., an

edge from S to T, again contradicting the definition of T. We conclude that $f(e') = 0$ for any such e'. Then the claim follows by by substituting $f(e) = c(e)$, $f(e') = 0$ into the equation of Remark 1.

$$v(f) = \sum_{\substack{e \text{ leaving } S, \\ \text{entering } T}} f(e) \; - \sum_{\substack{e' \text{ leaving } T, \\ \text{entering } S}} f(e') \; .$$

■

Two important corollaries follow from the proof of Ford-Fulkerson:

Corollary 1 (Max-Flow/Min-Cut) *The minimum cut value in a network is the same as the maximum flow value.*

Corollary 2 (Integral Flow) *If all edge capacities in a network are non-negative integers, then there exists an integral maximum flow.*

1.2 Run Time of the Ford-Fulkerson Algorithm[1]

Theorem 1 is predicated on the Ford-Fulkerson algorithm terminating, i.e. reaching a state where no more augmenting paths may be found in the residual network. The following lemma guarantees termination in the case of integral capacities.

Lemma 1 *If all edge capacities of N are integral, i.e. $c(e) \in \mathbb{N} \cup \{0\}$ $\forall e \in E$, Ford-Fulkerson terminates.*

Proof: Since the edge capacities are integral, the capacity of every edge in $R(N, f)$ is always integral. At each step, df is at least one; thus the value of flow f increases by at least one. Since $v(f) \leq \sum_{e \text{ leaving } s} c(e) < \infty$, $v(f)$ cannot increase indefinitely, hence the algorithm stops after a finite step. ■

Noting that finding an augmenting path through breadth-first search has a running time of $O(m)$, the proof of Lemma 1 implies that the running time of the Ford-Fulkerson algorithm is $O(mv(f^*))$ for integral c. The following example shows that this bound allows very slow convergence in extreme cases.

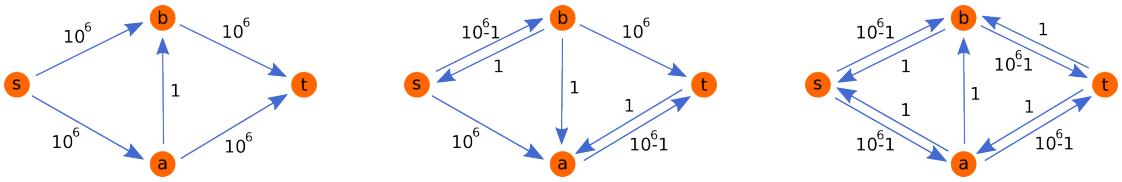
Example: Define a flow network on four nodes: a source s, a sink t and two nodes a and b connected as in Figure 2a. Consider the FF algorithm which on each iteration selects the augmenting flow path such that nodes a and b are both in the path. The first two iterations are illustrated in the figure. Each such iteration increases the value of the flow by 1 which implies that it will take $v(f^*) = 2 \cdot 10^6$ iterations until the algorithm terminates.

The $O(mv(f^*))$ running time bound is independent of how the augmenting paths are chosen in each step of the FF algorithm. We may use heuristics to more carefully select which augmenting path to use in each step; this approach allows run time bounds that are polynomial in the size of the input graph and do not depend on the capacities.

1.2.1 Widest Augmenting Path (WAP)

We consider an implementation of the Ford-Fulkerson algorithm in which we pick at every iteration the widest augmenting path in the residual network, where the width of a path in a capacitated network is the

[1]Some of the exposition in this section is borrowed from the Winter 2011 CS261 class notes written by Luca Trevisan. The material referenced may be found at `http://theory.stanford.edu/~trevisan/books/cs261.pdf` (lectures 9-11).



(a) Original Network. (b) Iteration 1. (c) Iteration 2.

Figure 2: Residual Networks after two iterations of the Ford Fulkerson algorithm picking an augmenting path s, a, b, t on the first iteration and s, b, a, t on the second.

minimum capacity of the edges in the path. In the network of the example above, the paths s, a, t and s, b, t have width 10^6, while the path s, a, b, t has width 1.

Lemma 2 *If N is a network with max-flow value $v(f^*)$, then there is a path from s to t of width $\geq v(f^*)/m$.*

Theorem 2 *Suppose that the network N has integral capacities. Then the WAP implementation of the Ford-Fulkerson algorithm runs in time $O(m^2 \log m \log v(f^*))$, where $m = |E|$, the number of edges in N.*

Proof: From the above lemma, we see that if we implement the Ford-Fulkerson algorithm with the widest path heuristic, then, after we have found augmenting paths, we have a solution such that, in the residual network, the optimum flow has cost at most $v(f^*) \cdot \left(1 - \frac{1}{2m}\right)^t$.

To see why, call v_i the cost of the flow found by the algorithm after i iterations, and r_i the optimum of the residual network after i iterations of the algorithm. Clearly we have $r_i = v(f^*) - v_i$. Lemma 2 tells us that at iteration $i+1$ we are going to find an augmenting path of width at least $r_i \cdot \frac{1}{2m}$ (the factor of 2 comes from the fact that each edge of N is possibly doubled in the residual network). This means that the cost of the flow at the end of the $(i+1)$-th iteration is going to be $v_{i+1} \geq v_i + r_i \cdot \frac{1}{2m}$, which means that the residual optimum is going to be

$$\begin{aligned} r_{i+1} &= v(f^*) - v_{i+1} \\ &\leq v(f^*) - v_i - r_i \cdot \frac{1}{2m} \\ &= r_i \left(1 - \frac{1}{2m}\right) \end{aligned}$$

The algorithm started with $v_0 = 0$ and $r_0 = v(f^*)$, so this inequality implies that $r_t \leq v(f^*) \cdot \left(1 - \frac{1}{2m}\right)^t$. For $t > 2m \log v(f^*)$, we may compute that $v(f^*) \cdot \left(1 - \frac{1}{2m}\right)^t < 1$. Since r_t is integer-valued, this means that $r_t = 0$ for $t > 2m \log v(f^*)$, i.e. the FF algorithm terminates within the first $1 + 2m \log v(f^*)$ iterations.

The widest augmenting path in a residual network may be computed using a dynamic programming approach in $O(m \log m)$ time. Then the total run time of WAP Ford-Fulkerson is $O(m^2 \log m \log v(f^*))$. ■

The running time of FF with WAP is polynomial in the bit description of the input since it takes at least $\log v(f^*)$ binary bits to encode the values of the capacities of our network. This is called a *weakly* polynomial time the algorithm. A *strongly* polynomial time algorithm is one that is polynomial in only the size of the problem which is given in terms of n and m. An example of such an algorithm for the max-flow problem is given in the next section.

1.2.2 Shortest Augmenting Path (SAP)

The shortest augmenting path implementation of the FF algorithm (also known as the Edmonds-Karp algorithm) chooses at each iteration the augmenting path containing the fewest number of edges.

Lemma 3 *If, at a certain iteration, the length of a shortest path from s to t in the residual network is l, then at every subsequent iteration it is $\geq l$. Furthermore, after at most m iterations, the distance from s to t must become $\geq l+1$.*

Theorem 3 *The SAP implementation of the Ford-Fulkerson algorithm runs in time $O(m^2n)$, where $m = |E|$, $n = |V|$ are the numbers of edges and vertices in N.*

Proof: The shortest augmenting path in the residual network at each iteration can be identified in $O(m)$ time using breadth-first search, and the above lemma gives us a way to upper bound the number of possible iterations. In particular, we note that as long as there is a path from s to t, the distance from s to t is at most $n-1$. Then the lemma tells us that after at most $m \cdot (n-1)$ iterations, s and t must become disconnected in the residual network, at which point the algorithm terminates. Thus the SAP Ford-Fulkerson algorithm runs in time $O(m^2n)$. ■

2 Applications of Network Flow

In this section we present a selection of problems that may be solved by applying the max-flow/min-cut theorem to an appropriately constructed network.

2.1 Baseball Elimination[2]

Suppose we have a set S of baseball teams and up to this point in the season, each team $x \in S$ has won w_x games. Through the remainder of the season every pair of teams $x, y \in S$ must still play g_{xy} games against each other. The question is whether a particular team z will be able to "win the division," i.e. whether there exists an outcome assignment for all remaining games such that z will finish with at least as many wins as any other team.

Example: Consider the (pre-1998) AL East division, where at some late date in the season we have the following win totals and games left to play:

Wins	Games to Play
$NYY : 93$	$NYY - BOS : 1$
$BOS : 89$	$NYY - TOR : 6$
$BAL : 86$	$NYY - BAL : 1$
$TOR : 88$	$BOS - BAL : 3$
	$TOR - BAL : 1$

Can Boston end up winning the division? In this case the answer is no; we establish a proof by noting that the maximum number of wins Boston can achieve is 93, while the set of teams $\{NYY, TOR\}$ must average $(93+88+6)/2 = 93.5 > 93$ wins, implying that one of those teams' win total must exceed 93.

[2] See Kleinberg and Tardos Section 7.12

In order to formulate the elimination question as a flow problem, we first assume that team z wins all of the games it has yet to play. Suppose that this leaves z with m wins at the season's end. If z is to win the division, we need a way to allocate all wins from the other games not involving z so that no team's win total exceeds m. We attempt to do this by solving the max-flow problem on the following network:

- Consider nodes s and t to be a "win source" and "win sink" respectively. Let $S' = S \setminus \{z\}$, and for each team $x \in S'$ include a node v_x. For every pair of teams $x, y \in S'$ include a node u_{xy}. So $V = \{s, t\} \cup \{v_x \mid x \in S'\} \cup \{u_{xy} \mid x, y \in S', x \neq y\}$.
- From the win source s to each node u_{xy} add the directed edge (s, u_{xy}) with capacity g_{xy}. This represents the fact that the games teams x and y play have at most g_{xy} wins to assign.
- From each node u_{xy}, add two edges (u_{xy}, v_x), (u_{xy}, v_y) with infinite capacity. Flow on these edges represents an assignment of wins from the x, y games to the teams.
- From each team node v_x, add the edge (v_x, t) with capacity $m - w_x$. This capacity ensures that v_x cannot receive enough flow from the game nodes (i.e. x cannot be assigned enough wins) to surpass team z in total wins.

Let $g^* = \sum_{x,y \in S', x \neq y} g_{xy}$, the total capacity leading out from the win source s. From the definition of the network the following claim is clear:

Claim 1 *The maximum flow in the network is g^* iff there is an assignment of wins so that no team's win total exceeds m. That is, team z is mathematically eliminated from contention iff the maximum flow is strictly less than g^*.*

Define an elimination proof set to be $T \subset S$ such that

$$\frac{1}{|T|}\left(\sum_{x \in T} w_x + \sum_{x,y \in T, x \neq y} g_{xy}\right) > m.$$

If T satisfies the above inequality, we may conclude that some team in T must end up with more than m wins; hence the existence of T gives a proof for z's elimination. The following lemma is left as an exercise to the reader.

Lemma 4 *Let (A, B) be the min-cut of the above network corresponding to the max-flow. If $c(A, B) < g^*$, then $T = \{x \in S \mid v_x \in A\}$ is an elimination proof set.*

2.2 Graph Connectivity[3]

An amateur graph theorist, in his scribblings, might invent the following two definitions of k-edge connectivity. G is k-edge connected if:

1. G remains connected after removing any $(k-1)$ edges.

   OR

2. There are at least k edge-disjoint paths between every pair of vertices in G.

[3]See Kleinberg and Tardos Section 7.6

Clearly if G satisfies definition 2 then it also satisfies definition 1. How about the other way around?

Theorem 4 *If a graph $G(V, E)$ remains connected after removing any $(k-1)$ edges then there are at least k edge-disjoint paths between every pair of vertices in G.*

Proof: Suppose that G satisfies definition 1 above, and let $s, t \in V$. Consider the directed version G' of G formed by replacing all edges $\{u, v\}$ with the two edges (u, v) and (v, u). Now let N be the network with G' as its underlying graph, s and t as source and sink, and all edges with capacity 1.

We claim that if we can show there are k edge-disjoint paths from s to t in G', then the same holds in G. To see this, note that we may turn all directed paths into undirected paths as long as two directed paths P_1 and P_2 don't use both directions (u, v) and (v, u) of an undirected edge $\{u, v\}$. If this is the case, however, we may simply have the two paths swap endings: define $\tilde{P}_1$ to be the path following P_1 from s to u and then P_2 from u to t, define $\tilde{P}_2$ to be the path following P_2 from s to v and then P_1 from v to t. Then if we consider $\tilde{P}_1$ and $\tilde{P}_2$ undirected, neither path uses $\{u, v\}$. All such conflicts can be inductively removed, proving the claim.

Now in order to find k edge-disjoint paths from s to t in G', we compute the maximum flow in N. The fact that G remains connected after removing any $(k-1)$ edges implies that the minimum cut in N has value at least k; thus the max-flow has value at least k. Since there must be an integral maximum flow in which the flow on every edge of N is $(0, 1)$-valued, we may exhibit k edge-disjoint paths from s to t in G' by greedily tracing paths from s to t along edges with flow 1. ■

2.3 Project Selection[4]

Suppose we are given:

- a set of projects $\{T_1, \ldots, T_n\}$,
- project T_i has a profit P_i which can be positive or negative,
- a project can be a prerequisite of another.

We would like to know what is the optimal subset of projects, i.e., the subset that satisfies the prerequisite requirement and has the maximum total profit?

Example: Consider a strip mine with layers of rocks and valuable minerals. Extracting a portion of a layer is a project that may be net profitable (valuable minerals) or net costly (hard rock). In order to reach minerals deep in the mine, we must remove all of the material covering it; each project is a prerequisite for those lying below it.

We formulate the project selection problem as a min-cut instance as follows. Define a network N on $G(V, E)$ in the following way. Let $V = \{s, t\} \cup \{T_1, T_2, \ldots T_n\}$. If $P_i \geq 0$ include the edge (s, T_i) with capacity P_i; if $P_i < 0$ include the edge $(T_i, t)$with capacity P_i. If T_j is a prerequisite of T_i include the edge (T_i, T_j) with capacity ∞.

Let (S, T) be the cut with the minimum value where $s \in S$ and $t \in T$. $S' = S \setminus \{s\}$ is a set of projects obeying the prerequisite rules, or else there would be an edge of capacity ∞ crossing the cut. This cannot occur because (S,T) is a min-cut and we can trivially exhibit cuts with finite value, e.g. $(\{s\}, V \setminus \{s\})$. Similarly, we can show that for any (A, B) with $c(A, B) < \infty$, $A' = A \setminus \{s\}$ is a feasible subset of tasks. Thus we aim

[4]See Kleinberg and Tardos Section 7.11

to show that finding the min-cut is equivalent to maximizing total profit, i.e. we show that the total profit of S' is greater than or equal to the total profit of A'. Let $P^+ = \sum_i P_i \mathbb{I}(P_i \geq 0)$ and $P^- = \sum_i P_i \mathbb{I}(P_i < 0)$, where $\mathbb{I}(\cdot)$ is the indicator function. It is easy to show that:

$$c(A,B) = P^+ - \sum_{P_i \in A} P_i \mathbb{I}(P_i \geq 0) - \left(P^- - \sum_{P_i \in A} P_i \mathbb{I}(P_i < 0) \right).$$

Thus $c(S,T) \leq c(A,B)$ implies that

$$P^+ - P^- - \left(\sum_{P_i \in S} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in S} P_i \mathbb{I}(P_i < 0) \right) \leq P^+ - P^- - \left(\sum_{P_i \in A} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in A} P_i \mathbb{I}(P_i < 0) \right)$$

or equivalently

$$\sum_{P_i \in S'} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in S'} P_i \mathbb{I}(P_i < 0) \geq \sum_{P_i \in A'} P_i \mathbb{I}(P_i \geq 0) - \sum_{P_i \in A'} P_i \mathbb{I}(P_i < 0)$$

Note that the left hand side is the total profit of S' and the right hand side is the total profit of A'. Thus we have shown that any other feasible solution has profit at most equal to the profit of S'.

2.4 Bipartite Matching[5]

A **bipartite graph** $G(V,E)$ is a graph whose vertex set can be partitioned as $V = X \cup Y$, with the property that every edge $e \in E$ has one end in X and the other end in Y.

A **matching** $M \subset E$ is a selection of edges such that for all $e, e' \in M$ either $e = e'$ or $e \cap e' = \emptyset$.

A **perfect matching** is a matching M such that every $v \in V$ belongs to some $e \in M$.

Theorem 5 *A bipartite graph $G(V = X \cup Y, E)$ with $|X| = |Y| = n$ has a perfect matching iff for all $S \subset X$, $|S| \leq |N(S)|$. $N(S)$ denotes the neighbors of S; $N(S) = \{v \in V \mid \{u,v\} \in E, u \in S\}$.*

Proof: The forward direction is trivial; the edges of a perfect matching provide a unique neighbor for each node in S. For the other direction, suppose for contradiction that G has no perfect matching but $|S| \leq |N(S)|$ holds for all $S \subset X$. We frame the matching problem in terms of network flow by turning G into a network N as follows:

- Direct all edges in G from X to Y and give them each capacity ∞.
- Add a node s and an edge (s,x) for each $x \in X$ with capacity 1.
- Add a node t and an edge (y,t) for each $y \in Y$ with capacity 1.

It is clear from the construction of N that G has a perfect matching iff the max-flow in N has value n. This means that G has a perfect matching iff the min-cut has value n; therefore our contradiction assumption implies that the min-cut has value strictly less than n. Consider a min-cut (A,B) and let $S = A \cap X$. Then all of the edges from s into $X \setminus S$ cross the cut; these edges have total capacity $|X \setminus S|$. Now, all neighbors of S in G must also lie in A, i.e $N(S) \subset A$, or else an edge of capacity ∞ would cross the cut. Then all of

[5] See Kleinberg and Tardos Section 7.5

the edges from the nodes of $N(S)$ to t cross the cut; these edges have total capacity $|N(S)|$. We have the equations:

$$|S| + |X \setminus S| = |X| = n$$
$$|X \setminus S| + |N(S)| \leq c(A, b) < n$$

which together imply that $|N(S)| < |S|$, a contradiction. ■