Mathematical Foundations of Computer Science

CS 499, Shanghai Jiaotong University, Dominik Scheder

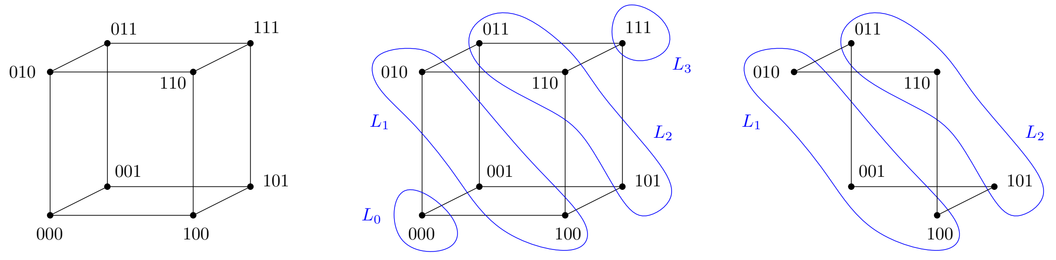
Spring 2019

11 Matchings and Network Flow

- Homework assignment published on Thursday, 2019-05-23.
- Submit questions and first solutions by Wednesday, 2019-05-29, 12:00.
- Submit final solution by Wednesday, 2019-06-05.

11.1 Matchings

Consider the Hamming cube $\{0,1\}^n$. We can view it as a graph H_n, where the vertex set is $\{0,1\}^n$ and two vertices x, y are connected by an edge if x and y differ in exactly one coordinate. Define the k^{th} layer to be $L_k := \{x \in \{0,1\}^n \mid |x|_1 = k\}$, where $|x|_1$ denotes the number of 1s in x. Note that the subgraph induced by layer k and layer $k+1$ is a bipartite graph $H_n[L_k \cup L_{k+1}]$. See the picture below for an illustration $(n = 3, k = 1)$:



Exercise 11.1. Let $0 \le k < n/2$. Show that the bipartite graph $H_n[L_k \cup L_{k+1}]$ has a matching of size $|L_k| = \binom{n}{k}$.

Solution. Degree of each vertex in L_k is $n-k$, and vertex in L_{k+1} is $k+1$. Because $0 \leq k < n/2$, then $n-k \geq k+1$. Also, for arbitrary two vertices in L_k, there is no overlap between edges that these two vertices can cover. By deleting a vertex in L_k, we always cover the maximal possible number of the remaining edges. Therefore, the minimum vertex cover is L_k. According to Konig Theorem, matching size is $|L_k|$.

Exercise 11.2. Let $G = (V, E)$ be a bipartite graph with left side L and right side R. Suppose G is d-regular (every vertex has degree d), so in particular $|L| = |R|$. Show that G has a perfect matching (that is, a matching M of size $|L|$).

Solution. Degree of each vertex is d. For arbitrary two vertices in L_k, there is no overlap between edges that these two vertices can cover. Consider the vertex cover problem, by always deleting a vertex in L (or R), we cover the maximal possible number of the remaining edges. Therefore, L and R are minimum vertex cover. According to Konig Theorem, maximum matching size is $|L|$. So we can find a matching of size $|L|$.

Exercise 11.3. Let G a d-regular bipartite graph. Show that the edges $E(G)$ can be partitioned into d perfect matchings. That is, there are matchings $M_1, \ldots, M_d \subseteq E(G)$ such that (1) $M_i \cap M_j = \emptyset$ for $1 \leq i < j \leq d$ and (2) $M_1 \cup M_2 \cup \cdots \cup M_d = E(G)$.

Solution. According to Exercise 11.2, we can find a perfect matching in a d-regular bipartite graph. After delete this perfect matching, the graph becomes a $(d-1)$-regular bipartite graph, where we can find another perfect matching. We keep doing this until there is no edge in the graph. Therefore, the edges $E(G)$ can be partitioned into d perfect matchings.

11.2 Networks with Vertex Capacities

Suppose we have a directed graph $G = (V, E)$ but instead of *edge capacities* we have *vertex capacities* $c : V \to \mathbb{R}$. Now a flow f should observe the *vertex capacity constraints*, i.e., the outflow from a vertex u should not exceed $c(u)$:

$$\forall u \in V : \sum_{v \in V, f(u,v) > 0} f(u, v) \leq c(u) \ .$$

Exercise 11.4. Consider networks with vertex capacities.

1. Show how to model networks with vertex capacities by networks with edge capacities. More precisely, show how to transform $G = (V, E, c)$ with $c : V \to \mathbb{R}^+$ into a network $G' = (V', E', c')$ with $c' : E' \to \mathbb{R}^+$ such that every s-t-flow f in G that respects the vertex capacities corresponds to an s-t-flow f' (of same value) in G' that respects edge capacities, and vice versa.

2. Draw a picture illustrating your solution.

3. Show that there is a polynomial time algorithm solving the following problem: Given a directed graph $G = (V, E)$ and two vertices $s, t \in V$. Are there k paths $p_1, \dots, p_k$, each from s to t, such that the paths are *internally vertex disjoint*? Here, internally vertex disjoint means that for $i \neq j$ the paths p_i, p_j share no vertices besides s and t.

Solution.

1. Transform G into G' in the following steps:

   (a) For each $v \in V$, split it into two vertices, v_{in} and v_{out}. Add them into V'. Add an edge (v_{in}, v_{out}) with capacity $c(v)$ into E'.

   (b) For each $(u, v) \in E$, add (u_{out}, v_{in}) with capacity ∞ into E'.

   Then the transformation is complete. Next we prove that every s-t-flow f in G corresponds to an s-t-flow f' of same value in G'.

   Proof. "$\Leftarrow$":An s-t-flow $f : V \times V \to \mathbb{R}$ in G satisfies that

$$\forall u \in V, \quad \sum_{(u,v)\in E} f(u, v) \leq c(u)$$
$$f(u, v) = -f(v, u) \quad (f(u, v) > 0 \text{ if } (u, v) \in E)$$
$$\sum_{v\in V} f(u, v) = 0, \ \forall u \in V \setminus \{s, t\}$$

   For corresponding f', we only need to show the capacity constraint is satisfied as the other two constraints are just copied as they have the same value. Combine two of the constraints together we have

$$f'(u_{in}, u_{out}) = \sum_{(v_{out},u_{in})\in E'} f'(v_{out}, u_{in}) = \sum_{(v,u)\in E} f(v, u) = \sum_{(u,v)\in E} f(u, v) \leq c(u) = c'(u_{in}, u_{out})$$

And for other edges in E', it is trivial to show $f'(u,v) \le c'(u,v) = \infty$.

Therefore the corresponding $f' : V' \times V' \to \mathbb{R}$ in G' satisfies that $\forall u, v \in V'$, $f'(u,v) \le c'(u,v)$, which is indeed a flow w.r.t edge capacities.

"$\Rightarrow$":An s-t-flow $f' : V' \times V' \to \mathbb{R}$ in G' satisfies that

$$\forall u, v \in V', \quad f'(u,v) \le c'(u,v)$$
$$f'(u,v) = -f'(v,u) \quad (f'(u,v) > 0 \text{ if } (u,v) \in E')$$
$$\sum_{v \in V'} f'(u,v) = 0, \ \forall u \in V' \setminus \{s,t\}$$

Likewise, we only need to show the corresponding $f : V \times V \to \mathbb{R}$ in G satisfies the capacity constraint. Actually we have

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(u_{out},v_{in})\in E'} f'(u_{out}, v_{in}) = f'(u_{in}, u_{out}) \le c'(u_{in}, u_{out}) = c(u)$$

Therefore f is a flow w.r.t vertex capacities. $\square$

2. Figure 1 shows an example of the transform.

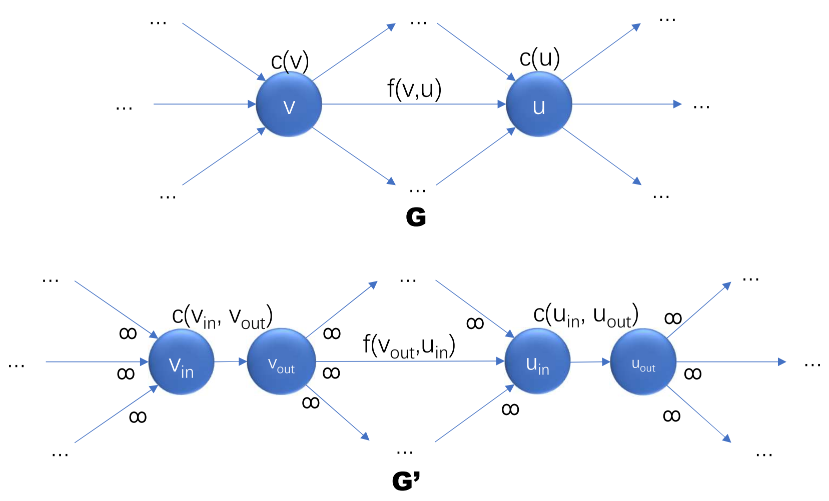


Figure 1: An illustration of the transformation

3. The algorithm requires modification to the original graph $G = (V, E)$. And it is similar to the process we transform the network with vertex capacities into a new net work with edge capacities.

   First, for each $v \in V$, split it into two vertices, v_{in} and v_{out} and add an edge (v_{in}, v_{out}) with capacity 1. Then for each $(u, v) \in E$, convert it into (u_{out}, v_{in}) with also capacity 1. Then we run Ford-Fulkerson or Edmonds-karp to compute ths max $s - t$ flow. If the result is greater than or equal to k, the original answer is yes, there are k internally vertex disjoint paths from s to t in G. Else the answer is no. See the pseudo code in Algorithm 1.

Algorithm 1 Algorithm to check whether k internally vertex disjoint $s - t$ paths exist

```
 1: procedure k VERTEX-DISJOINT s − t PATHS($G = (V, E), s, t \in V$)
 2:     Create a new empty graph $G' = (V', E')$
 3:     for all vertex $v \in V$ do
 4:         $V' \leftarrow V' \cup \{v_{in}, v_{out}\}$
 5:         $E' \leftarrow E' \cup \{(v_{in}, v_{out})\}$
 6:         $c(v_{in}, v_{out}) = 1$
 7:     end for
 8:     for all edge $(u, v) \in E$ do
 9:         $E' \leftarrow E' \cup \{(u_{out}, v_{in})\}$
10:         $c(u_{out}, v_{in}) = 1$
11:     end for
12:     Compute max $s_{out} - t_{in}$ flow in $G'$, save result as $f$
13:     if $f \geq k$ then
14:         return True
15:     else
16:         return False
17:     end if
18: end procedure
```

   For the construction of the new graph G', the time complexity is O($|V| + |E|$). For computing the max flow, it cost polynomial time. Thus in all this is a polynomial time algorithm.

   For correctness of this algorithm, it is easy to see that one unit of flow in G' corresponds to a path in G. Since the capacity between (v_{in}, v_{out})

is restricted to 1, for each pair of two vertices, it is crossed by at most one unit of flow, which corresponds to that in the original graph, no vertex is shared by two paths. Therefore the value of max flow in G' equals to the number of internally vertex disjoint paths in G, w.r.t s, t.

Exercise 11.5. Let H_n be the n-dimensional Hamming cube. For $i < n/2$ consider L_i and L_{n-i}. Note that $|L_i| = \binom{n}{i} = \binom{n}{n-i} = L_{n-i}$, so the L_i and L_{n-i} have the same size. Show that there are $\binom{n}{i}$ paths $p_1, p_2, \ldots, p_{\binom{n}{i}}$ in H_n such that (i) each path p starts in L_i and ends in L_{n-i}; (ii) two different paths p, p' do not share any vertices.

Proof. We can extract all vertices and edges between (including) L_i and L_{n-i}, and additionally add a source s and a sink t, connect all vertices in L_i to s and all vertices in L_{n-i} to t, to form a new graph G, where all vertices have capacity 1 except for s, t.

According to our statement in 11.4, if the maximum $s - t$ flow in the new graph is exactly $\binom{n}{i}$, then there exists $\binom{n}{i}$ internally vertex disjoint $s - t$ paths in that graph. Further, since all such paths are vertex disjoint, this would mean that in H_n, correspondingly each path starts in L_i and ends in L_{n-i}, with no vertex shared by two paths.

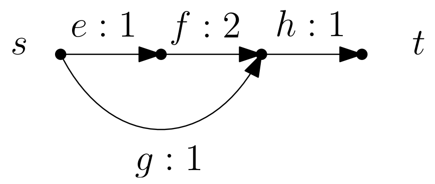
Remember in 11.1 we proved that the bipartite graph $H_n[L_k \cup L_{k+1}]$ has a matching of size $|L_k| = \binom{n}{k}$ for $0 \leq k < n/2$, which is a perfect matching since all vertices are covered. From antoher point of view, this tells us that the max flow between L_k and L_{k+1} with respect to vertex capacities is $|L_k|$. With k becoming closer to $n/2$, $|L_k| = \binom{n}{k}$ monotonically increases, and max flow between two layers increases. For $k > n/2$, the case is symmetric as $n - k < n/2$, $L_{n-k} = L_k = \binom{n}{k}$.

Therefore the minimum max flow between two adjacent layers in G is $|L_i| = |L_{n-i}| = \binom{n}{i}$, and flow between nonadjacent layers is 0, which restircts the max $s - t$ flow in G to be exactly $\binom{n}{i}$. □

11.3 Always, Sometimes, or Never Full

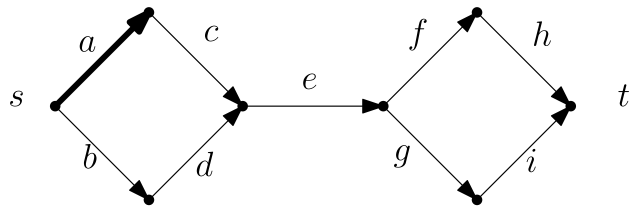
Let (G, s, t, c) be a flow network, $G = (V, E)$. A directed edge $e = (u, v)$ is called always full if $f(e) = c(e)$ for every maximum flow; it is called sometimes full if $f(e) = c(e)$ for some but not all maximum flows; it is called never full if $f(e) < c(e)$ for all maximum flows.

Let $(S, V \setminus S)$ be a cut. That is, $s \in S, t \in V \setminus S$. We say the edge $e = (u, v)$ is crossing the cut if $u \in S$ and $v \in V \setminus S$. We say e is always crossing if it crosses every minimum cut; sometimes crossing if it crosses some, but not all minimum cuts; never crossing if it crosses no minimum cut. For example, look at this flow network:



Example network: the edges e, g are sometimes full and never crossing; f is never full and never crossing; h is always full and always crossing.

Exercise 11.6. Consider this network:



The fat edge a has capcity 2, all other edges have capacity 1.

1. Indicate which edges are (i) always full, (ii) sometimes full, (iii) never full.
2. Indicate which edges are (i) always crossing, (ii) sometimes crossing, (iii) never crossing.

Proof.

1. Always full: e

   Sometimes full: b, c, d, f, g, h, i

   Never full: a

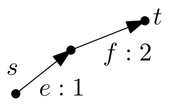
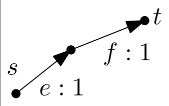
2. Always crossing: e

   Somtimes crossing: no one

   Never crossing: a, b, c, d, f, g, h, i

□

Exercise 11.7. An edge e can be (x) always full, (y) sometimes full, (z) never full; it can be (x') always crossing, (y') sometimes crossing, (z') never crossing. So there are nine possible combinations: (xx') always full and always crossing, (xy') always full and sometimes crossing, and so on. Or are there? Maybe some possibilities are impossible. Let's draw a table:

The edge e is:	x: always full	y: sometimes full	z: never full
x': always crossing		Possible or impossible?	Possible or impossible?
y': sometimes crossing		Possible or impossible?	Possible or impossible?
z': never crossing	Possible or impossible?	Possible or impossible?	Possible or impossible?

The nine possible cases, some of which are maybe impossible.

The two very simple flow networks in the table already show that (xx') and (yy') are possible; that is, it is possible to be always full and always crossing, and it is possible to be always full and sometimes crossing. Fill out the table! That is, for each of the remaining seven cases, find out whether it is possible or not. If it is possible, draw a (simple) network showing that it is possible; if impossible, give a proof of this fact.

Proof. (xz') is impossible. Since the maximum flow equals to the minimum cut, if an edge is always full then it is possible to be part of some minimum cut.

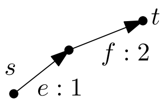
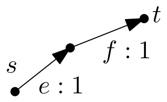
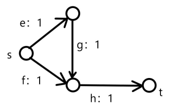
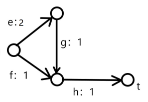
(yx') and (yy') are impossible. If a network has several different minimum cut, then the edges in these minimum cut should be full in the maximum flow.

(zx') and (zy') are impossible. By the definition, if an edge is not full, it cannot be a part of the minimum cut.

$\square$

11.4 Multi-Commodity Flow

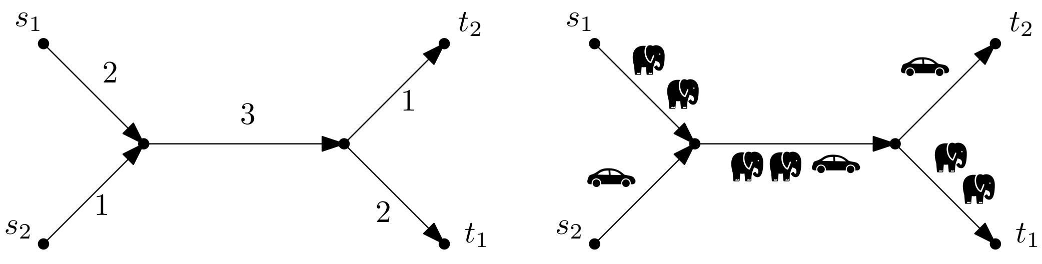
In class, we discussed the Multi-Commodity Flow problem. Formally, a multi-commodity network is given by a directed graph $G = (V, E)$, a capacity function $c : E \to \mathbb{R}^+$, and sources $\vec{s} = (s_1, \ldots, s_k)$ and sinks $\vec{t} = (t_1, \ldots, t_k)$ in V. A *multi-commodity flow* in $(G, c, \vec{s}, \vec{t})$ is a tuple $(f_1, \ldots, f_k)$ where each f_i is an s_i-t_i-flow in (G, c, s_i, t_i), that is, f_i is individually a flow, satisfying

The edge e is:	x: always full	y: sometimes full	z: never full
x': always crossing		Impossible	Impossible
y': sometimes crossing		Impossible	Impossible
z': never crossing	Impossible		

flow conservation constraints at all vertices except s_i and t_i; the *value* of f_i, $\text{val}(f_i)$, is the outflow at s_i, as usual. Furthermore, $\sum_{i=1}^{k} f_i(e) \leq c(e)$ for all edges $e \in E$. Think of each f_i as being a way to route units of good i from s_i to t_i.

The *Maximum Multi-Commodity Flow Problem* (Max-MCF) asks for a multi-commodity flow in the network maximizing the total value, i.e., $\text{val}(f_1)+\cdots+\text{val}(f_k)$.

In the *Feasibility Multi-Commodity Flow Problem* (F-MCF), we are additionally given *demands* $d_1, \ldots, d_k$, and we want to decide whether there is a multi-commodity flow $(f_1, \ldots, f_k)$ with $\text{val}(f_i) = d_i$. If there is such a multi-commodity flow, we say the instance of F-MCF is *feasible*, otherwise we say it is *infeasible*.

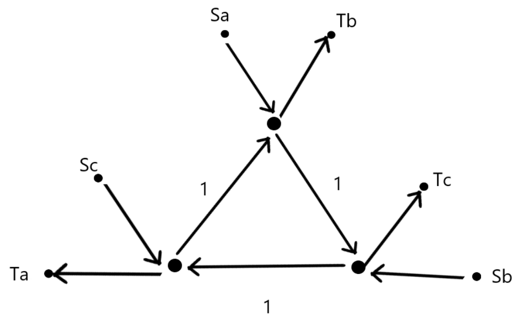


A multi-commodity flow routing two elephants from s_1 to t_1 and one car from s_2 to t_2. Note that the two flows share the middle edge; also, all flow values and all capacities are integers.

For each of Max-MCF and F-MCF we can define the *integer* version: Max-IMCF and F-IMCF. These are the same problems as above, but we additionally require that all capacities, demands, and flow values be integers.

Exercise 11.8. Find a multi-commodity flow network with integer capacities such that Max-MCF is larger than Max-IMCF. That is, to achieve the maximum possible flow, it is necessary to use non-integral flows.

Proof. All capacities are 1. All demands are 1, too. The Max Commodity Flow is 1.5. □



Exercise 11.9. Find a multi-commodity flow network with integer capacities and integer demands such that the F-MCF problem is feasible but the F-IMCF problem is infeasible. That is, it is possible to satisfy all the demands, but not all flows must be integer.