Mat 3770
Network Flows

Spring 2014

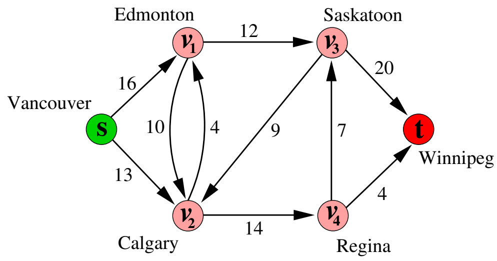
4.2 Network Flows

- A directed graph can model a **flow network** where some material (e.g., widgets, current, ...) is **produced** or enters the network at a **source** and is **consumed** at a **sink**.
- Production and consumption are at a **steady rate**, which is the same for both.
- The **flow** of the material at any point in the system is the rate at which the material moves through it.

Modeling

- Flow networks can be used to model:
  - liquids through pipe
  - parts through an assembly line
  - current through electrical networks
  - info through communication networks
- Each directed edge is a **conduit** for the material.
- Each conduit has a stated **capacity** given as a maximum rate at which the material can flow through the conduit. (e.g., 200 barrels of oil per hour.)

A Network Flow Example



A flow network for the Lucky Duck Puck factory, located in Vancouver, with warehouse in Winnipeg.
Each edge is labeled with its capacity.

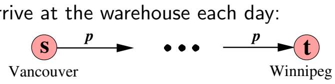
from: *Introduction to Algorithms*, by Cormen, Leiserson, & Rivest

Example Details

- The Lucky Duck Company has a **factory** (source s) in Vancouver that manufactures hockey pucks.
- They have a **warehouse** (sink t) in Winnipeg that stores them.
- They lease space on trucks from another firm to ship the pucks from the factory to the warehouse — with **capacity** $c(u, v)$ crates per day between each pair of cities u and v.

Goal: determine p, the largest number of crates per day that can be shipped, and then produce this amount — there's no sense in producing more pucks than they can ship to their warehouse.

- The rate at which pucks are shipped along any truck route is a **flow**.
- **Maximum flow** determines p, the maximum number of crates per day that can be shipped.
- The pucks leave the factory at the rate of p crates per day, and p crates must arrive at the warehouse each day:



- Shipping time is not a concern, only the flow of p crates/day.
- **Capacity constraints** are given by the restriction that the flow $f(u, v)$ from city u to city v be at most $c(u, v)$ crates per day.
- In a **steady state**, the number of crates entering and the number leaving an intermediate city must be equal.

Flow Conservation

- Vertices are conduit **junctions**. Other than the **source** and **sink**, material flows through the vertices **without collecting** in them.
- Hence, the rate at which material **enters** a vertex must **equal** the rate at which it **leaves** the vertex.
- This property is called **flow conservation**, and is similar in concept to Kirchhoff's Current Law concerning electrical current.

Maximum–flow

The **Maximum–flow** problem is the simplest problem concerning flow networks:

What is the greatest rate at which material can be shipped from source to sink without violating any capacity constraints?

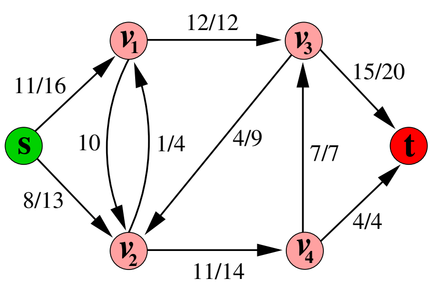
Assumptions

A **flow network**, G = (V, E), is a directed graph in which each edge (u, v) $\in$ E has a non–negative **capacity** c(u, v) ≥ 0.

If (u, v) $\notin$ E, we assume c(u, v) = 0.

$\forall x \in$ E, In(x) and Out(x) are the edges into and out of vertex x.

The integer c(e) associated with edge e is a capacity or upper bound.



A flow f with value $|f| = 19$.
Only positive net flows (crates shipped) are shown.
If $f(u, v) > 0$, edge (u, v) is labeled by $f(u, v)/c(u, v)$.
If $f(u, v) \leq 0$, edge (u, v) is labeled only by its capacity.

Tucker: a–z (Source to Sink) Flow ϕ

- An **a–z flow** ϕ in a directed network N is an integer–valued function ϕ defined on each edge e.
- $\phi(e)$ is the flow in e, together with a **source** vertex a and a **sink** vertex z satisfying the following three conditions:

  1. **Capacity Constraint:** $0 \leq \phi(e) \leq$ c(e)
     We don't want *backflow*, nor to exceed any edge's capacity
  2. $\phi(e) = 0$ if $e \in$ IN(a) or $e \in$ OUT(z)
     We want the flow to go from source to sink, not vice–versa
  3. **Flow Conservation:**
     For x $\neq$ a or z, $\sum_{e \in IN(x)} \phi(e) = \sum_{e \in OUT(x)} \phi(e)$
     For every vertex other than the source and sink, the flow into and out of that vertex must be equal

Flow Networks

A **flow** in G is a real–valued function $f : V \times V \to \Re$ that satisfies the following three properties:

1. **Capacity constraint**: the flow along an edge cannot exceed its capacity:
$$\forall\ u, v, \in V, \quad f(u, v) \leq c(u, v)$$
2. **Skew symmetry**: the flow from a vertex u to a vertex v is the negative of the flow in the reverse direction:
$$\forall\ u, v \in V, \quad f(u, v) = -f(v, u)$$
3. **Flow conservation**: the net flow of a vertex (other than the source or sink) is 0:
$$\forall\ u \in V - \{s, t\}, \quad \sum_{v \in V} f(u, v) = 0$$

- The quantity $f(u, v)$, which can be positive, negative, or zero, is called the **flow** from vertex u to vertex v.
- The **value** of a flow f is defined as:
$$|f| \quad = \quad \sum_{v \in V} f(s, v) \quad = \quad \sum_{v \in V} f(v, t).$$
I.e., the total flow **out of the source** or **into the sink**.
- In the **maximum–flow problem**, we are given a flow network G with source s and sink t, and we wish to find a flow of maximum value from s to t.

1. (By skew symmetry) The flow from a vertex to itself is 0, since for all $u \in V$, we have $f(u, u) = -f(u, u)$
2. (By skew symmetry) We can rewrite the flow–conservation property as the total flow into a vertex is 0:
$$\forall\ v \in V - \{s, t\}, \quad \sum_{u \in V} f(u, v) = 0$$

Flow In Equals Flow Out

1. The total positive flow **entering** a vertex v is defined by
$$\sum_{u \in V, f(u,v) > 0} f(u, v)$$
2. The total positive flow **leaving** a vertex v is defined by
$$\sum_{u \in V, f(v,u) > 0} f(v, u)$$
3. The positive net flow entering a vertex (other than the source or sink) must equal the positive net flow leaving the vertex.

There can be no flow between u and v
if there is no edge between them.

If there is no edge between u and v, i.e., $(u, v) \notin E$ and $(v, u) \notin E$, then:

- If there is no edge, then the capacity is zero.
And by the Capacity Constraint, the flows must be ≤ 0.
$\mathrm{c(u,\ v) = c(v,\ u) = 0 \quad \Rightarrow \quad f(u,\ v) \leq 0,\ f(v,\ u) \leq 0}$
- By skew symmetry, the flow must be zero.
$\mathrm{f(u,\ v) = -f(v,\ u) \quad \Rightarrow \quad f(u,\ v) = f(v,\ u) = 0}$

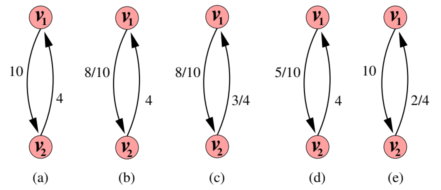
Cancellation

- **Cancellation** allows us to represent the shipments between two cities by a positive flow along at most one of the two edges between the corresponding vertices.
- If there is zero or negative flow from one vertex to another, no shipments need be made in that direction.
- Any situation in which pucks are shipped in both directions between two cities can be transformed using **cancellation** into an **equivalent** situation in which pucks are shipped **only** in the direction of **positive flow**.
- No constraints are violated since the net flow between the two vertices is the same.

Cancellation in General

- Suppose edge (u, v) has flow value $f(u, v)$
- Now, increase the flow on the on edge (v, u) by some amount d
- Mathematically, this operation must decrease $f(u, v)$ by d
- Conceptually, we can think of these d units as canceling d units of flow that are already on edge (u, v)

Cancellation Example

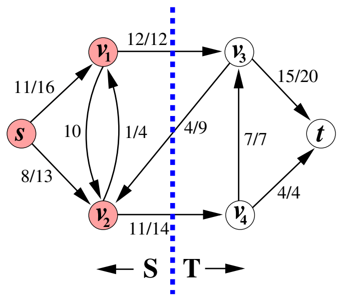


(a) Vertices v_1 and v_2, with $c(v_1, v_2) = 10$ and $c(v_2, v_1) = 4$
(b) Net flow when 8 crates per day are shipped from v_1 to v_2.
(c) An additional shipment of 3 crates per day from v_2 to v_1.
(d) By canceling flow going in opposite directions, we can represent the situation in (c) with positive flow in one direction only.
(e) Another 7 crates per day shipped from v_2 to v_1 results in a net of 2 crates per day from v_2 to v_1.

Flow Cuts

- A **cut(S, T)** of flow network G = (V, E), is a partition of V into S and T = V - S, such that **source** ∈ S and **sink** ∈ T.
- If f is a flow, then the **net flow** across the $cut(S, T)$ is defined to be $f(S, T)$.
- The **capacity** of the $cut(S, T)$ is $c(S, T)$.

Cut Example



cut({a, v_1, v_2}, {v_3, v_4, z}) has **net flow**:
$f(v_1, v_3) + f(v_2, v_3) + F(v_2, v_4) = 12 - 4 + 11 = 19.$
And, its **capacity** is: $c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26.$

Ford–Fulkerson Method

Each iteration will increase the flow until

the maximum is reached

```
Ford-Fulkerson Method (G, s, t)
   initialize flow f to 0
   while there exists an augmenting path p
      augment flow f along p
   return f
```


Ford–Fulkerson Solution

The Ford–Fulkerson Method for solving the maximum–flow problem depends on three key concepts:

1. residual networks
2. augmenting paths
3. cuts

Residual Networks

Given a flow network and a flow, the **residual network** consists of edges that can accommodate more net flow.

- Suppose we have a flow network G = (V, E), with source s and sink t.
- Let f be a flow in G; consider a pair of vertices u, v $\in$ V.
- The amount of **additional** flow we can push from u to v before exceeding the capacity $c(u, v)$ is the **residual capacity** of (u, v) given by:

$$c_f(u, v) = c(u, v) - f(u, v)$$

- For example, if $c(u, v) = 16$ and $f(u, v) = 11$, we can ship $c_f(u, v) = 5$ more units before we exceed capacity.

- When the **flow** is **negative**, the **residual capacity** is **greater than the capacity**

  E.g., $c(u, v) = 16$, $f(u, v) =$ -4, so $c_f(u, v) = 20$

- This can be interpreted to mean:
  1. There is a flow of 4 units from v $\rightarrow$ u, which we can cancel by pushing a flow of 4 units from u $\rightarrow$ v.
  2. We can then push another 16 units from u $\rightarrow$ v before violating the capacity constraint on edge (u, v).
  3. We have thus pushed an additional 20 units of flow, starting with a flow $f(u, v) = -4$, before reaching the capacity constraint.

- Given a flow network G= (V, E) and a flow f, the **residual network** of G induced by f is $G_f = (V, E_f)$, where:

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0$$

  I.e., the Residual network consists of edges that can accommodate more net flow.

- Each edge of the residual network (**residual edge**), can admit a strictly positive new flow.
- Notice (u, v) may be a residual edge in E_f even if it was not an edge in E — i.e., it could be the case that $E_f \not\subset E$.

- Because an edge (u, v) can appear in a residual network only if at least one of (u, v) and (v, u) appears in the original network, we have the bound: $|E_f| \leq 2|E|$.
- Observation: the residual network G_f is itself a flow network with capacities given by c_f.
- **Lemma**. Let G = (V, E) be a flow network with source s and sink t, and let f be a flow in G. Let G_f be the residual network of G induced by f, and let f' be a flow in G_f. Then the flow sum $f + f'$ is a flow in G with value:

$$|f + f'| = |f| + |f'|$$

  This lemma shows how a flow in a residual network relates to a flow in the original flow network.

Augmenting Paths

- Given a flow network G = (V, E) and a flow f, an **augmenting path** p is a simple path from s to t in the residual network G_f.
- By the definition of the residual network, each edge (u, v) on an augmenting path admits some additional positive flow from u to v without violating the capacity constraint on the edge.
- The maximum amount of flow we can ship along the edges of an augmenting path p is the **residual capacity** of p given by:

$$c_f(p) = min\{c_f(u, v) : (u, v) \text{ is on } p\}$$

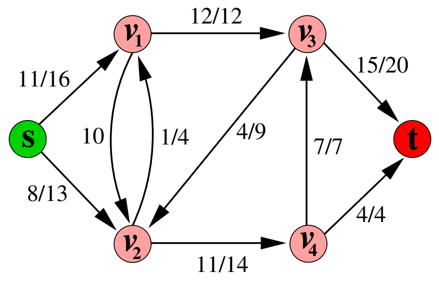
Lemma. Let G = (V, E) be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Define a function $f_p : V \times V \rightarrow \Re$ by:

$$f_p(u, v) = \begin{cases} c_f(p) & : \text{ if } (u, v) \text{ is on } p \\ -c_f(p) & : \text{ if } (v, u) \text{ is on } p \\ 0 & : \text{ otherwise} \end{cases}$$

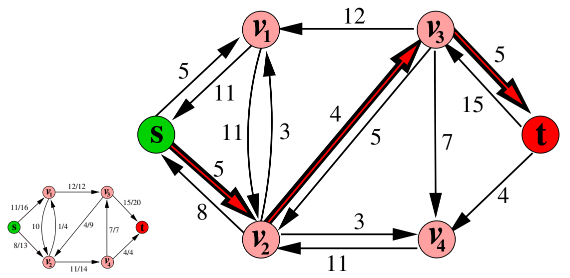
- If we add f_p to f, we get another flow in G whose value is closer to the maximum.
- **Corollary**. Let G = (V, E) be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Let f_p be defined as in the previous lemma. Define a function $f' : V \times V \to \Re$ by $f' = f + f_p$. Then f' is a flow in G with value:

$$|f'| = |f| + |f_p| > |f|$$

Flow Network G and Flow F

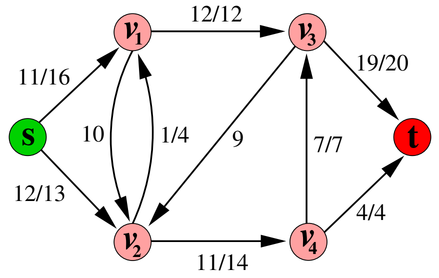


Residual network G_f



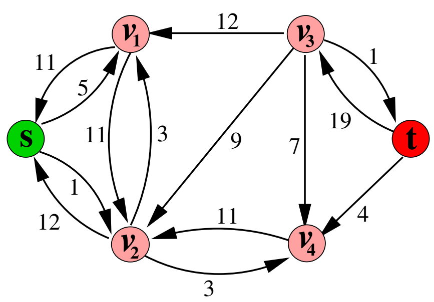
Residual network G_f with augmenting path p indicated; its residual capacity is $c_f(p) = c(v_2, v_3) = 4$

Resulting Flow After Augmenting



Resulting flow in G from augmenting path p by its residual capacity

New Residual Network



Residual network induced by the flow in (c)

Cuts of Flow Networks

The Ford–Fulkerson method repeatedly augments the flow along augmenting paths until a maximum flow has been found.

The max–flow min–cut theorem tells us a flow is maximum

IFF

its residual network contains no augmenting path.

The Max–flow Min–cut Theorem

- **Lemma**. Let f be a flow in a flow network G with source s and sink t, and let (S, T) be a cut of G. Then the net flow across (S, T) is f(S, T) = $|f|$.
- **Corollary**. The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.
- **Max–flow min–cut Theorem**. If f is a flow in a flow network G = (V, E) with with source s and sink t, then the following conditions are **equivalent**:
  1. f is a maximum flow in G
  2. The residual network G_f contains no augmenting paths
  3. $|f|$ = c(S, T) for some cut (S, T) of G

The Basic Ford–Fulkerson Algorithm

- Each iteration of the Ford–Fulkerson method,
  finds *any* augmenting path p and augments flow f along p by the residual capacity c_f(p)
- The Ford–Fulkerson method computes the maximum flow in a graph G = (V, E) by updating the net flow $f(u, v)$ between each pair of vertices that are connected by an edge.

  If u and v are not connected by an edge, assume $f(u, v) = 0$
- Assume the capacity from u to v is provided by a constant–time function $c(u, v)$ with:

$$c(u, v) = 0 \textit{ if } (u, v) \notin E$$

- The residual capacity is computed as $c_f(u, v) = c(u, v) - f(u, v)$, represented by $c(p)$ in the algorithm.


```
Ford-Fulkerson (G, s, t)
1  for each edge (u, v) in E[G]
2      f[u, v] = 0
3      f[v, u] = 0

4  while there exists a path(s, t) in the
          residual network Gf
5    cf(p) = min{cf(u, v): (u, v) in p}
6    for each edge (u, v) in p
7       f[u, v] = f[u, v] + cf(p)
8       f[v, u] = -f[u,v]
```

Lines 1 – 3 initialize the flow f to 0

The while loop repeatedly finds an augmenting path p in G_f, augments flow f along p by the residual capacity $c_f(p)$

When no augmenting path exists, flow f is a maximum flow.

Time Analysis — Assuming Integer Capacities

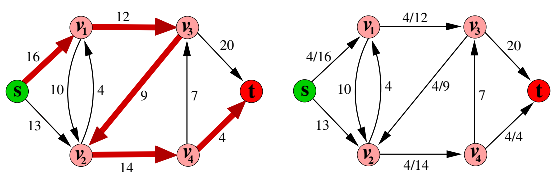
- Initialization, lines 1 – 3: $\Theta(|E|)$
- The while loop of lines 4 – 8 is executed at most $O(|f^*|)$ times, where f^* is the maximum flow found by the algorithm, since the flow value increases by at least one unit in each iteration.
- Let $m(p)$ = time to find a path with minimum residual capacity, and
  max = the maximum number of edges in such a path.
  Then the overall run time is:

$$O(E + |f^*|(m(p) + max))$$

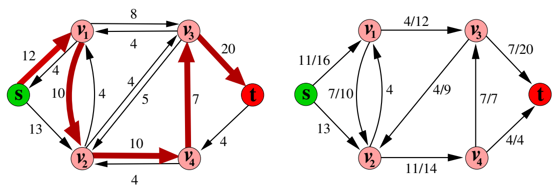
Successive iterations of the Ford–Fulkerson algorithm are shown on the next set of slides.

On each slide, the first graph (on left) shows the residual network G_f with an augmenting path p.

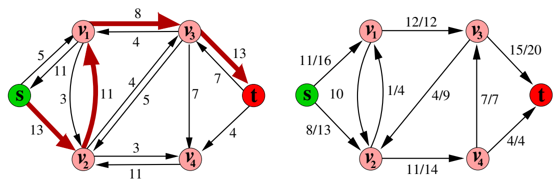
The second graph (right) shows the new flow f that results from adding f_p to f.



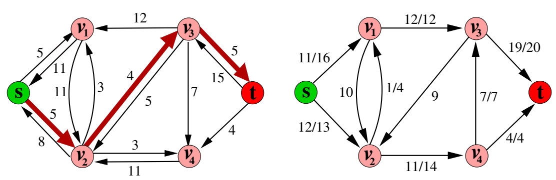
(a) The residual network (left) is the input network G
On right is the new, resulting flow



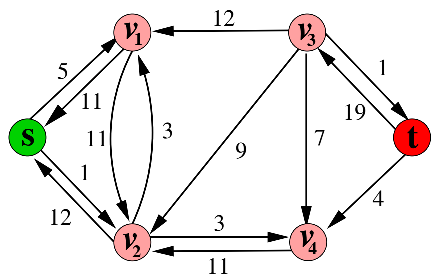
(b) The next augmenting path, with new flow



(c) The next augmenting path with, new flow

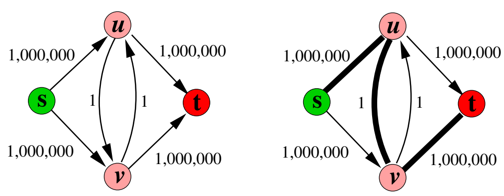


(d) The last augmenting path, with new flow



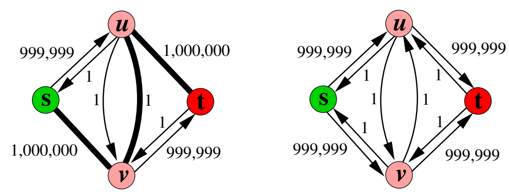
(e) No augmenting paths remain;
flow shown in (d) is maximum flow

Worst Case Example



(a) A flow network for which the algorithm can take $\Theta(E|f^*|)$ time

An augmenting path with residual capacity 1 is shown.



(b) Another augmenting path with residual capacity 1, with the resulting residual network.

Eventually, the maximum flow of $|f^*| = 2{,}000{,}000$ will be reached.

Edmonds–Karp Algorithm

- The bound on the Ford–Fulkerson algorithm can be improved if the computation of the augmenting path p (in line 4) is implemented with a breadth–first search.
- That is, if the augmenting path is a *shortest* path from s to t in the residual network, where each edge has unit distance (weight).
- The algorithm will then run in $O(VE^2)$ time.

4.3 Algorithmic Matching

- Some combinatorial problems can easily be cast as maximum–flow problems.
- One example: finding a maximum matching in a bipartite graph.
- The problem: Given an undirected graph G = (V, E), a **matching** is a subset of edges $M \subseteq E \ni \forall v \in V$, at most one edge of M is incident on v.

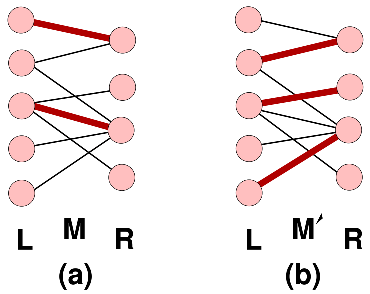
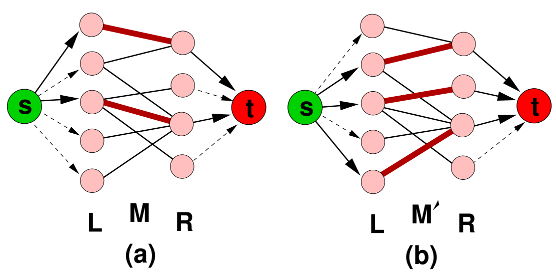


Fig 1. Bipartite graph G = (V, E), vertex partition $V = L \cup R$
(a) A matching with cardinality 2
(b) A maximum matching with cardinality 3

- Vertex $v \in V$ is said to be **matched** by matching M if some edge in M is incident on v; otherwise v is **unmatched**.
- A **maximum matching** is a matching of maximum cardinality: a matching $M \ni$ for any matching M', we have $|M| \geq |M'|$.
- We are interested in finding maximum matchings in bipartite graphs.
- Assume the vertex set can be partitioned into $V = L \cup R$, where L and R are disjoint and all edges in E go between L and R.

Recast as a Flow Network Problem



A Practical Application

- One (of many) practical application: matching a **set** L **of machines** with a **set** R **of tasks** to be performed simultaneously.
- The edge $< u, v > \in E$ indicates a particular machine $u \in L$ is capable of performing a particular task $v \in R$
- A maximum matching provides work for as many machines as possible.

Finding a Maximum Bipartite Matching

- We can use the Ford–Fulkerson method to find a maximum matching in an undirected bipartite graph, G + (V, E), in time polynomial in $|V|$ and $|E|$.
- The trick is to construct a flow network in which flows correspond to matchings.

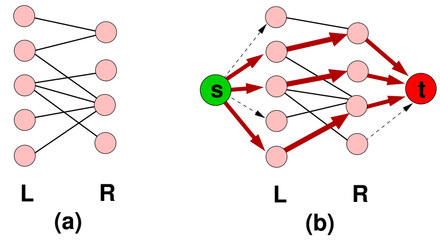


Fig. 2. The flow network corresponding to a bipartite graph.
(a) Bipartite graph with vertex partition from Fig. 1.
(b) Corresponding flow network with a maximum flow shown.

Each edge has unit capacity. Large arrows have a flow of 1, and all other edges carry no flow. The large arrows from L to R correspond to those in a max matching of the bipartite graph.

The **corresponding flow network** $G' = (V', E')$ for the bipartite graph G is defined as:

- Let the source s and sink t be new vertices not in V, and let $V' = V \sup\{s, t\}$.
- If the vertex partition of G is $V = L \cup R$, the directed edges of G' are given by:
$$\begin{aligned} E' = &\{< s, u > \ : \ u \in L\} \cup \\ &\{< u, v > \ : \ U \in L, \ v \in R, \text{ and } < u, v > \ \in \ E\} \ \cup \\ &\{< v, t > \ : \ v \in R\} \end{aligned}$$
- I.e., the edges from the source node to nodes in L, the original edges from L to R, and the edges from the nodes in R to the sink node.
- We then assign unit capacity to each of these edges in E'

- A matching in G corresponds directly to a flow in the corresponding flow network G'.
- A flow f on a flow network $G = (V, E)$ is **integer–valued** if $f(u, v)$ is an integer $\forall < u, v > \in V \times V$.
- **Lemma**. Let $G = (V, E)$ be a bipartite graph with vertex partition $V = l \cup R$, and let $G' = (V', E')$ be its corresponding flow network.

  If M is a matching in G, then there is an integer–valued flow f in G' with value $|f| = |M|$.

  Conversely, if f is an integer–valued flow in G', then there is a matching M in G with cardinality $|M| = |f|$.

- Intuitively, a maximum matching in a bipartite graph corresponds to a maximum flow in its corresponding flow network.
- We can compute a maximum matching in a bipartite graph by finding a maximum–flow in its flow network.
- The only possible problem: the maximum–flow algorithm might return a flow which consists of non–integral amounts (which would **not** lead to a good match).

- The following theorem shows that if we use the Ford–Fulkerson method, this difficulty cannot arise.
- **Theorem 1**. (**Integrality theorem**). If the capacity function c takes on only integral values, then the maximum flow f produced by the Ford–Fulkerson method has the property $|f|$ is integer–valued.

  Moreover, for all vertices u and v, the value of $f(u, v)$ is an integer.

  Proof is by induction on the number of iterations.

Corollary (to Lemma 1). The cardinality of a maximum matching in a bipartite graph G is the value of a maximum flow in its corresponding flow network G'.

Proof is by contradiction.