Chap 13. Network Flows

Yih-Lang Li (李毅郎)
Computer Science Department
National Chiao Tung University, Taiwan

The sources of most figure images are from the course slides (Graph Theory) of Prof. Gross

Outline

- Flows and Cuts in Networks
- Solving the Maximum-Flow Problem
- Flows and Connectivity
- Matchings, Transversals, and Vertex Covers

13.1 Flows and Cuts in Networks

- **DEFINITION**: *A **single source-single sink network** is a connected digraph that has a distinguished vertex called the **source**, with nonzero out-degree, and a distinguished vertex called the **sink**, with nonzero in-degree.*
- **DEFINITION**: *A **capacitated network** is a connected digraph such that each arc is assigned a nonnegative weight cap(e), called the **capacity** of arc e.*
- **TERMINOLOGY**: *A single source-single sink network with source s and sink (or target) t is often referred to as an **s-t network**. All of the networks discussed in this chapter are assumed to be capacitated s-t networks.*
- **DEFINITION**: Let *v* be a vertex in a digraph *N*.
  - ✓ The ***out-set*** of *v*, denoted *Out(v)*, is the set of all arcs that are directed from vertex *v*. That is, $Out(v) = \{e \in E_N | tail(e) = v\}$
  - ✓ The ***in-set*** of *v*, denoted *In(v)*, is the set of all arcs that are directed to vertex *v*. That is, $In(v) = \{e \in E_N | head(e) = v\}$

Flows and Cuts in Networks

- **NOTATION**: For any two vertex subsets X and Y of digraph N, let $\langle X, Y \rangle$ denote the set of all arcs in N that are directed from a vertex in X to a vertex in Y. That is,

$$\langle X, Y \rangle = \{e \in E_N \mid tail(e) \in X \text{ and } head(e) \in Y\}$$

- **Example 13.1.1:** Fig 13.1.1 shows a 5-vetex capacitated s-t network.
  If

$$X = \{x, v\} \text{ and } Y = \{w, t\}$$

  then

$$\langle X, Y \rangle = \{xw, vt\}$$

  The only element in arc set $\langle Y, X \rangle$ is the arc wv.

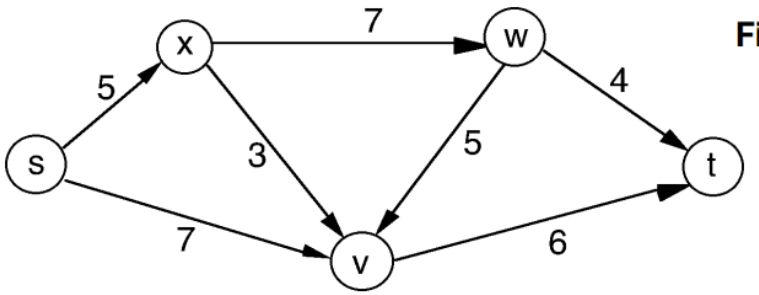


Figure 13.1.1 A 5-vertex capacitated network with source s and sink t.

Feasible Flows

- **DEFINITION**: Let *N* be a capacitated *s*-*t* network. A (***feasible***) ***flow*** *f* in *N* is a function

$$f : E_N \rightarrow R^+$$

that assigns a nonnegative real number *f*(*e*) to each arc such that:

$$f(e) \leq cap(e) \quad (\forall e \in E_N) \qquad (capacity\ constraint)$$

$$\sum_{e \in In(v)} f(e) = \sum_{e \in Out(v)} f(e) \quad (\forall v \in N - \{s,t\}) \qquad (conservation)$$

- **NOTATION**: To distinguish visually between the flow and the capacity of an arc, we adopt this convention in drawings:
  - ✓ When an arc is labeled with both capacity and flow, the capacity appears in bold and to the left of the flow.

Feasible Flows

- **Example 13.1.2.** Figure 13.1.2 shows a feasible flow for the capacitated network of Example 13.1.1. Notice that
  - ✓ The total amount of flow leaving source *s* equals 6, and also, the net flow entering sink *t* equals 6.

  The conservation of flow at every internal vertex in the network is intuitively consistent with this phenomenon. Later in this section, we establish the general result that overflow from the source equals inflow to the sink.

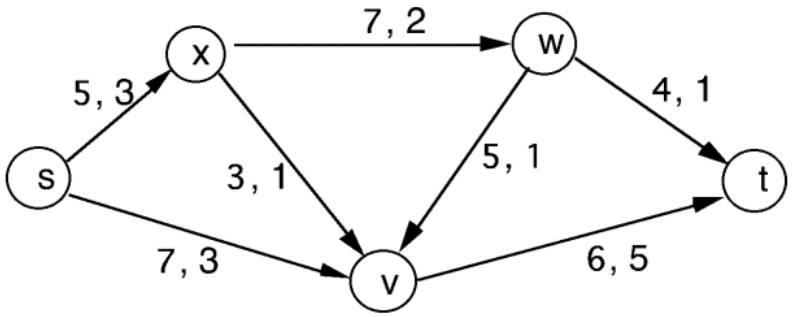


Fig 13.1.2 A flow for the example network.

- **DEFINITION**: The ***value of flow*** *f* in a capacitated network, denoted *val*(*f*), is the net flow leaving the source *s*, that is

$$val(f) = \sum_{e \in Out(s)} f(e) - \sum_{e \in In(s)} f(e)$$

- **DEFINITION**: *A **maximum flow** f* in a capacitated network N is a flow in N having the maximum value, i.e., val(f) ≤ val(f*), for every flow in N.*

Cuts in *s-t* Networks

- **DEFINITION**: *Let N be an s-t network, and let V_s and V_t form a partition of V_N such that source $s \in V_s$ and the sink $t \in V_t$. Then the set of all arcs that are directed from a vertex in set V_s to a vertex set V_t is called an s-t* ***cut*** *of network N and is denoted $<V_s, V_t>$.*
- **Remark:** Notice that the arc sets *Out*(*s*) and *In*(*t*) for an *s-t* network *N* are the *s-t* cuts $<\{s\}, V_N - \{s\}>$ and $<V_N - \{t\}, \{t\}>$, respectively.
- **Example 13.1.3:** Figure 13.1.3 portrays the arc sets *Out*(*s*) and *In*(*t*) as *s-t* cuts, where

$$Out(s) = \langle \{s\}, \{x, v, w, t\} \rangle \quad \text{and} \quad In(t) = \langle \{s, x, v, w\}, \{t\} \rangle$$

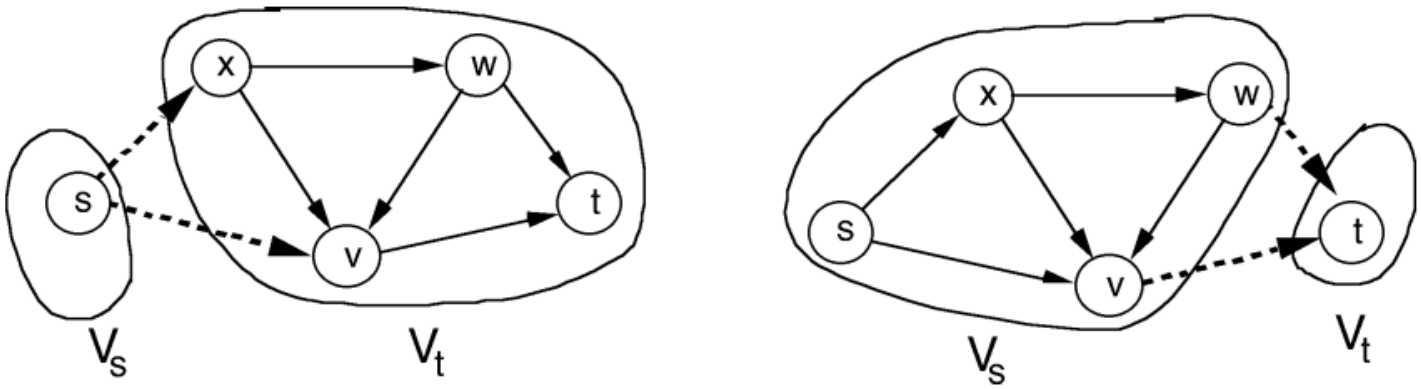


Fig 13.1.3 **Arc sets $Out(s)$ and $In(t)$ shown as *s-t* cuts.**

Cuts in *s-t* Networks

- **Example 13.1.4.** A more general *s-t* cut $<V_s, V_t>$, where $V_s = \{s, x, v\}$ and $V_t = \{w, t\}$

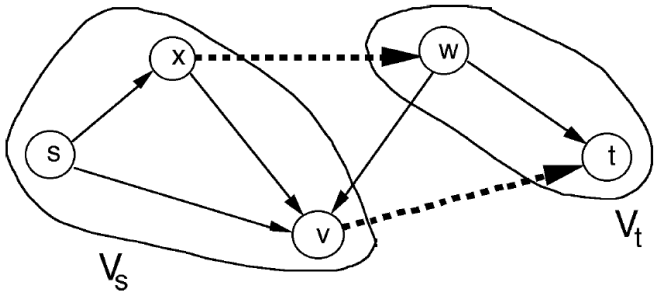


Fig 13.1.4 The s-t cut $\langle\{s, x, v\}, \{w, t\}\rangle$.

- **Proposition 13.1.1.** *Let $<V_s, V_t>$ be an s-t cut of network N. Then every directed s-t path in N contains at least one arc in $<V_s, V_t>$.*
  - ✓ Let $P = <s = v_0, v_1, v_2, \ldots, v_l = t>$ be the sequence of a directed *s-t* path in network *N*.
  - ✓ Since $s \in V_s$ and the $t \in V_t$, there must be a first vertex v_j on this path that is in set V_t. Then the arc from vertex v_{j-1} to v_j is in $<V_s, V_t>$.

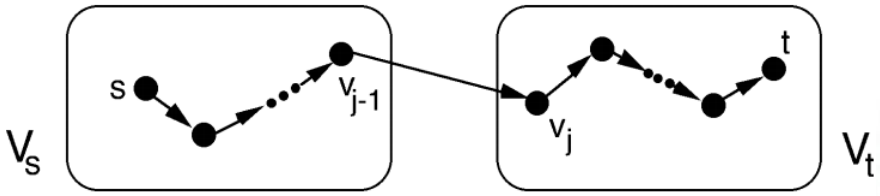


Fig 13.1.5 A directed s-t path with an arc in $\langle V_s, V_t\rangle$.

Relationship Between Flows and Cuts

- Similar to viewing the set *Out*(*s*) of arcs directed from source *s* as the *s*-*t* cut $\langle\{s\}, V_N - \{s\}\rangle$ the set *In*(*s*) may be regarded as the set of "backward" arcs relative to this cut, named the arc set $\langle V_N - \{s\}, \{s\}\rangle$.
  From this perspective, the definition of *val*(*f*) may be rewritten as

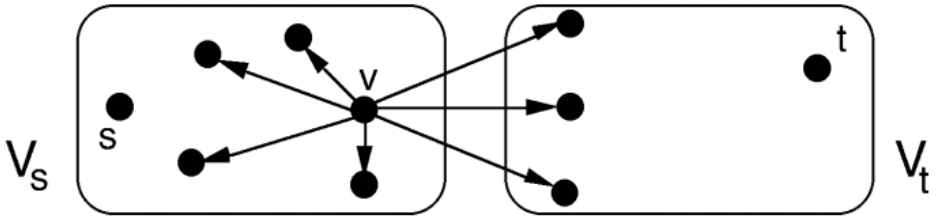
$$val(f) = \sum_{e \in \langle \{s\}, V_N - \{s\} \rangle} f(e) - \sum_{e \in \langle V_N - \{s\}, \{s\} \rangle} f(e)$$

  In other words, the value of any flow equals the total flow across the cut $\langle\{s\}, V_N - \{s\}\rangle$ minus the flow across the arcs of $\langle V_N - \{s\}, \{s\}\rangle$.

- **Lemma 13.1.2:** Let $\langle V_s, V_t\rangle$ *be an s-t cut of network N*
  Then

$$\bigcup_{v \in V_s} Out(v) = \langle V_s, V_s\rangle \cup \langle V_s, V_t\rangle \quad \text{and}$$

$$\bigcup_{v \in V_s} In(v) = \langle V_s, V_s\rangle \cup \langle V_t, V_s\rangle$$



  - ✓ For any vertex $v \in V_s$, each arc directed from *v* is either in $\langle V_s, V_s\rangle$ or in $\langle V_s, V_t\rangle$ (Figure 13.1.6 illustrates for a vertex *v*, the partition of *Out*(*v*) into a 4-element subset of $\langle V_s, V_s\rangle$ and a 3-element subset of $\langle V_s, V_t\rangle$.) Similarly, each arc directed to vetex *v* is either in $\langle V_s, V_s\rangle$ or in $\langle V_t, V_s\rangle$.

Relationship Between Flows and Cuts

Proposition 13.1.3. Let f be a *flow* in a *s-t* network N, and let $\langle V_s, V_t \rangle$ be any *s-t* cut of N. Then,

$$val(f) = \sum_{e \in \langle V_s, V_t \rangle} f(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e)$$

- By definition, $val(f) = \sum_{e \in Out(s)} f(e) - \sum_{e \in In(s)} f(e)$, and by conservation of flow, $\sum_{e \in Out(v)} f(e) - \sum_{e \in In(v)} f(e) = 0$, for every $v \in V_s$ other than s.

$$val(f) = \sum_{v \in V_s} \left(\sum_{e \in Out(v)} f(e) - \sum_{e \in In(v)} f(e) \right) = \sum_{v \in V_s} \sum_{e \in Out(v)} f(e) - \sum_{v \in V_s} \sum_{e \in In(v)} f(e)$$

- By Lemma 13.1.2,

$$\sum_{v \in V_s} \sum_{e \in Out(v)} f(e) = \sum_{e \in \langle V_s, V_s \rangle} f(e) + \sum_{e \in \langle V_s, V_t \rangle} f(e)$$

and

$$\sum_{v \in V_s} \sum_{e \in In(v)} f(e) = \sum_{e \in \langle V_s, V_s \rangle} f(e) + \sum_{e \in \langle V_t, V_s \rangle} f(e)$$

The result follows by substitution.

Relationship Between Flows and Cuts

- **Example 13.1.5.** The flow *f* and the cut $\{s, x, v\},\{w, t\}$, shown in Figure 13.1.7, illustrate Proposition 13.1.3.

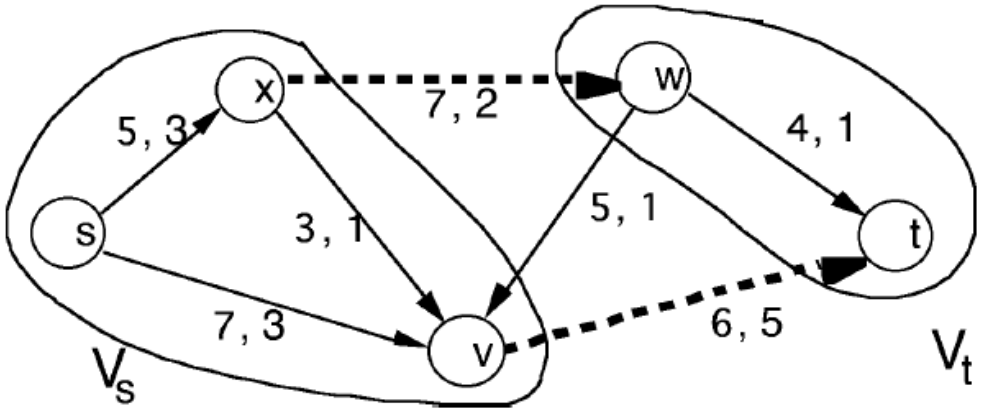


Figure 13.1.7

$$6 = val(f) = \sum_{e\in\langle\{s,x,v\},\{w,t\}\rangle} f(e) - \sum_{e\in\langle\{w,t\},\{s,x,v\}\rangle} f(e) = 7 - 1$$

- The next result confirms what was apparent earlier from intuition, namely, that the net flow out of source *s* equals the net flow into sink *t*.

Relationship Between Flows and Cuts

- **Corollary 13.1.4** *Let f be a flow in an s-t network. Then*

$$val(f) = \sum_{e \in In(t)} f(e) - \sum_{e \in Out(t)} f(e)$$

  - ✓ Apply Prop 13.1.3 to the *s-t* cut $In(t) = \langle V_N - \{t\}, \{t\} \rangle$

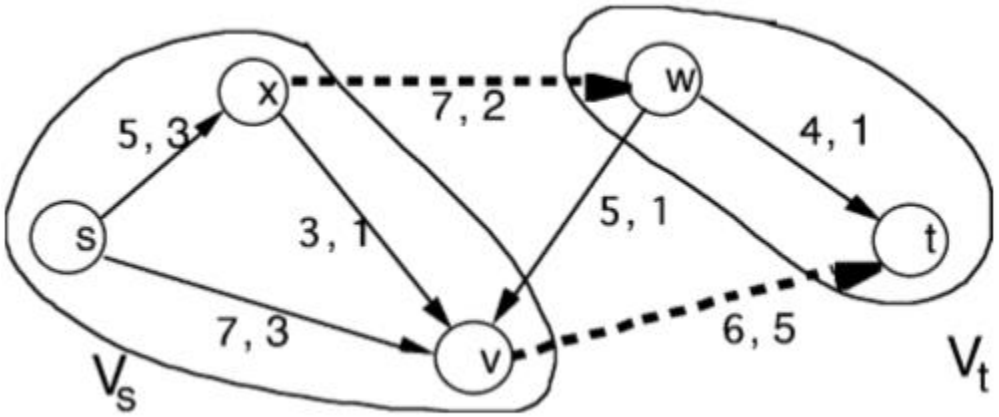
- **DEFINITION**: The ***capacity of a cut*** $\langle V_s, V_t \rangle$, denoted cap $\langle V_s, V_t \rangle$, is the sum of the capacities of the arcs in cut $\langle V_s, V_t \rangle$. That is,

$$cap\langle V_s, V_t \rangle = \sum_{e \in \langle V_s, V_t \rangle} cap(e)$$

- **DEFINITION**: A ***minimum cut*** of a network *N* is a cut with the minimum capacity.

Relationship Between Flows and Cuts

- **Example 13.1.6.** The capacity of the cut shown in Figure 13.1.7 is 13, and the cut $\langle \{s, x, v, w\}, \{t\} \rangle$ with capacity 10, is the only minimum cut.



Max-Flow Problem and Min-Cut Problem

- **Proposition 13.1.5** *let f be any flow in an s-t network N, and let* $\langle V_s, V_t \rangle$ *be any s-t cut. Then* $val\,(f) \leq cap\,\langle V_s, V_t \rangle$
  - ✓ The following chain of inequalities, which begins with the assertion of Proposition 13.1.3, establishes the result.

$$
\begin{aligned}
val(f) &= \sum_{e \in \langle V_s, V_t \rangle} f(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e) \qquad \text{(Prop 13.1.3)} \\
&\leq \sum_{e \in \langle V_s, V_t \rangle} cap(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e) \quad \text{(cap. constraint)} \\
&= cap\langle V_s, V_t \rangle - \sum_{e \in \langle V_t, V_s \rangle} f(e) \quad \text{(def of } cap\langle V_s, V_t \rangle\text{)} \\
&\leq cap\langle V_s, V_t \rangle \quad \text{(since each } f(e) \text{ is nonnegative)} \qquad \diamondsuit
\end{aligned}
$$

Max-Flow Problem and Min-Cut Problem

- **Corollary 13.1.6 [Weak Duality for Flows].** *Let f* be a maximum flow in an s-t network N, and let K* be a minimum s-t cut in N. Then val(f*) ≤ cap (K*)*
  - ✓ This is an immediate consequence of Prop 13.1.5.
- **Corollary 13.1.7 [Certificate of Optimality for Flows].** *Let f be a flow in an s-t network N and K an s-t cut, and suppose that val(f) = cap(K). Then flow f is a maximum flow in network N, and cut K is a minimum cut.*
  - ✓ Let $\hat{f}$ be any feasible flow in network *N*. The maximality of flow *f* is implied by Prop 13.1.5 and the premise, as follows.

$$val(\hat{f}) \leq cap(K) = val(f)$$

- The minimality of cut *K* can be established with a similar argument.

Max-Flow Problem and Min-Cut Problem

- **Example 13.1.7.** The flow for the example network shown in Figure 13.1.8 has value 10, which also is the capacity of the *s*-*t* cut $\langle \{s, x, v, w\}, \{t\} \rangle$. By Corollary 13.1.7, both the flow and the cut are optimal for their respective problems.

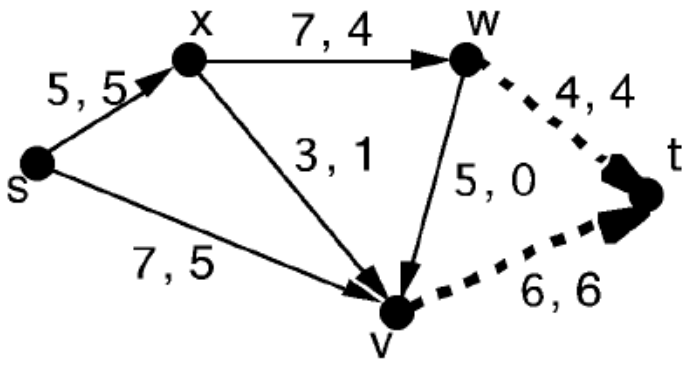


Fig 13.1.8 A maximum flow and minimum cut.

Max-Flow Problem and Min-Cut Problem

- The final result of this section is used in the next section to establish the correctness of a classical algorithm for finding the maximum flow in a network.
- **Corollary 13.1.8.** *Let $\langle V_s, V_t \rangle$ be an s-t cut in a network N, and suppose that f is a flow such that*

$$f(e) = \begin{cases} cap(e) & \text{if } e \in \langle V_s, V_t \rangle \\ 0 & \text{if } e \in \langle V_t, V_s \rangle \end{cases}$$

Then f is a maximum flow in N, and $\langle V_s, V_t \rangle$ is a minimum cut.

13.2 Solving the Maximum-Flow Problem

- The basic idea of the algorithm presented in this section, originated by Ford and Fulkerson [FoFu62], is to increase the flow in a network iteratively until it cannot be increased any further. Each increase is based on a suitably chosen alternating sequence of vertices and arcs, called an *augmenting flow path*. Before introducing the most general such path, we consider the simplest and most obvious one.

Using Directed Paths to Augment Flow

- Suppose that f is a flow in a capacitated s-t network N, and suppose that there exists a directed s-t path $P = \langle\, s, e_1, e_2, \ldots, e_k, t \,\rangle$ in N, such that $f(e_i) < cap(e_i)$, for $i = 1, \ldots, k$
- Then considering arc capacities only, the flow on each arc e_i can be increased by as much as $cap(e_i) - f(e_i)$
- But to maintain the conservation-of-flow property at each of the vertices v_i, the increases on all of arcs of path P must be equal. Thus, if Δp denotes this increase, then the largest possible value for Δp is $min\{cap(e_i) - f(e_i)\}$

Using Directed Paths to Augment Flow

- **Example 13.2.1.** In the example network shown on the left in Figure 13.2.1, the value of the current flow is 6. Consider the directed *s*-*t* path $P = \langle s, x, w, t \rangle$
- The flow in each arc of path P can increase by $\Delta p = 2$, and the resulting flow, which has value 8, is shown on the right.

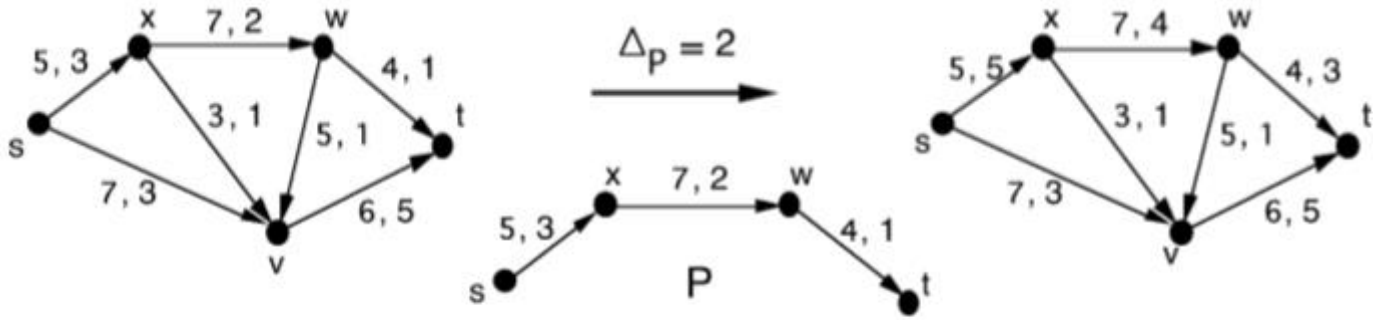


Figure 13.2.1 Using directed path P to increase the flow from 6 to 8.

Using Directed Paths to Augment Flow

- Using the directed path $\langle s,v,t \rangle$, the flow can then be increased to 9. The resulting flow is shown in Figure 13.2.2. At this point, the flow cannot be increased any further along *directed s-t* paths, because each such path must use either the arc directed from source *s* to vertex *x* or the one from vertex *v* to sink *t*, and both these arcs have flow at capacity.

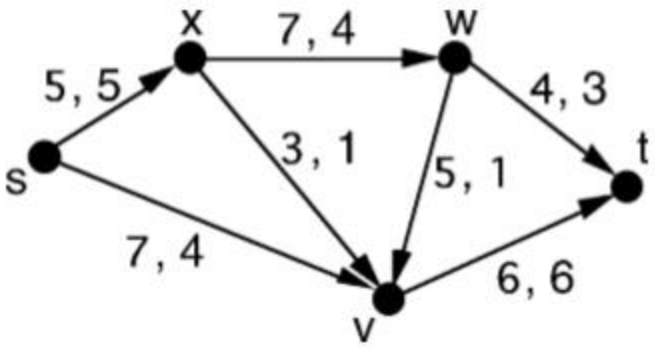


Figure 13.2.2 Existing flow cannot be increased along a directed path.

f-Augmenting Paths

- **DEFINITION**: An *s*-*t* ***quasi-path*** in a network N is an alternating seq $\langle s = v_0, e_1, v_1, \ldots, v_{k-1}, e_k, v_k = t \rangle$ that forms an *s*-*t* path in the underlying graph of N. The arc e_i is a ***forward arc*** if directed from vertex v_{i-1} to vertex v_i, or a ***backward arc*** if directed from v_i to v_{i-1}.

- **Example 13.2.2.** the *s*-*t* quasi-path in Fig 13.2.3, arcs *a* and *b* are backward, and the three other arcs are forward.

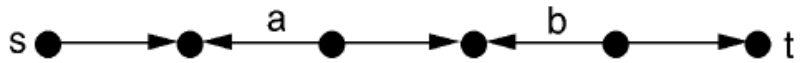


Fig 13.2.3 A quasi-path with two backward arcs.

f-Augmenting Paths

- **DEFINITION**: Let *f* be a flow in an *s*-*t* network *N*. An ***f*-*augmenting* path** *Q* is an *s*-*t* quasi-path in *N*, such that the flow on each forward arc can be increased and the flow on each backward arc can be decreased.
- Thus, for each arc *e* on an *f*-augmenting path *Q*,

$$\begin{aligned} f(e) &< cap(e), && \text{if } e \text{ is a forward arc} \\ f(e) &> 0, && \text{if } e \text{ is a backward arc} \end{aligned}$$

- **NOTATION:** For each arc *e* on a given *f*-augmenting path, let Δ_e be the quantity given by

$$\Delta_e = \begin{cases} cap(e) - f(e), & \text{if } e \text{ is a forward arc} \\ f(e), & \text{if } e \text{ is a backward arc} \end{cases}$$

- **TERMINOLOGY:** The quantity Δ_e is called the ***slack on arc*** *e*. Its value on a forward arc is the largest possible increase in the flow, and on a backward arc, the largest possible decrease in the flow, *disregarding conservation of flow*.

f-Augmenting Paths

- **Remark:** Conservation of flow requires that the change in the flow in the arcs of an augmenting flow path be of equal magnitude. Thus, the maximum allowable in the flow on an arc of quasi-path Q is Δ_Q , *where*

$$\Delta_Q = \min_{e \in Q} \{\Delta_e\}$$

- Notice that this definition of Δ_Q coincides with that of Δ_P, defined earlier, whenever the quasi-path Q is a directed path.

f-Augmenting Paths

- **Example 13.2.3.** For the example network in Figure 13.2.4, the current flow f has value 9, and the quasi-path $Q = \langle s, x, w, t \rangle$ is an f-augmenting path, with $\Delta_Q = 1$.

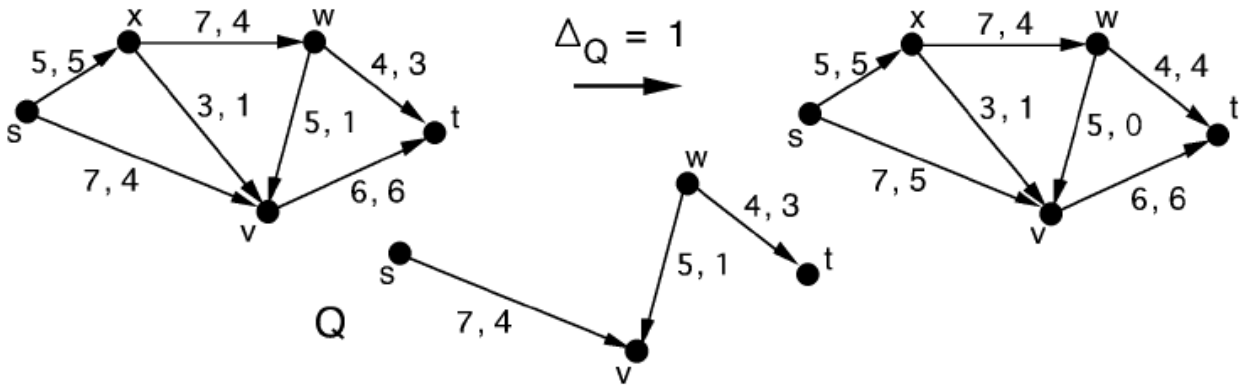


Figure 13.2.4 Using an f-augmenting path Q to increase the flow by $\Delta_Q = 1$.

f-Augmenting Paths

- **Proposition 13.2.1 [Flow augmentation]** Let f be a flow in a network N, and let Q be an f-augmenting path with minimum slack Δ_Q on its arcs. Then the augmented flow $\hat{f}$ given by

$$\hat{f}(e) = \begin{cases} f(e) + \Delta_Q, & \text{if } e \text{ is a forward arc of } Q \\ f(e) - \Delta_Q, & \text{if } e \text{ is a backward arc of } Q \\ f(e), & \text{otherwise} \end{cases}$$

is a feasible flow in network N, and $val(\hat{f}) = val(f) + \Delta_Q$

  - ✓ Clearly, $0 \le \hat{f}(e) \le cap(e)$, by def of Δ_Q . The only vertices through which the net flow may have changed lie on the augmenting path Q. Thus, to verify that $\hat{f}$ satisfies conservation of flow, only the internal vertices of Q need to be checked.
  - ✓ For a given vertex v on augmenting path Q, the two arcs of Q that are incident on v are configured in one of the four ways shown in Fig 13.2.5. In each case, net flow at vertex v does not change, thereby preserving the conservation-of-flow property.

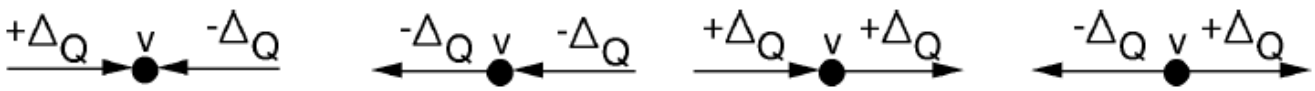


Figure 13.2.5 The four possibilities for an internal vertex v of a quasi-path.

f-Augmenting Paths

- ✓ It remains to be shown that the flow has increased by Δ_Q . **The only arc incident on source *s* whose flow has changed is the first arc e_1 of augmenting path *Q*.** If e_1 is a forward arc, then $\hat{f}(e_1) = f(e_1) + \Delta_Q$, and if e_1 is a backward arc, then $\hat{f}(e_1) = f(e_1) - \Delta_Q$. In either case,

$$val(\hat{f}) = \sum_{e \in Out(s)} \hat{f}(e) - \sum_{e \in In(s)} \hat{f}(e) = \Delta_Q + val(f) \quad \diamondsuit$$

- **Corollary 13.2.2** Let *f* be an integer-valued flow in a network whose arc capacities are integer. Then the flow that results from each successive augmentation will be integer-valued.

Max-Flow Min-Cut Theorem

- **Theorem 13.2.3 [Characterization of Max Flow]** Let *f* be a flow in a network *N*, then *f* is a max flow in network *N* iff there does not exist *f*-augmenting path in *N*.
  - ✓ *Necessity*(→) Suppose that *f* is a max flow in network *N*. Then by Proposition 13.2.1, there is no *f*-augmenting path.
  - ✓ *Sufficiency*(←) Suppose that there does not exist an *f*-augmenting path in network *N*. Consider the collection of all quasi-paths that start at vertex *s* and have the following property: each forward arc on the quasi-path has positive slack (i.e., it can be increased) and each backward arc on the quasi-path has positive flow (i.e., it can be decreased). Let V_s be the union of the vertex-sets of these quasi-paths.

    Since there is no *f*-augmenting path, it follows that sink $\notin V_s$. Let $V_t = V_N - V_s$. Then $<V_s , V_t>$ is an *s*-*t* cut of network *N*. Moreover, by definition of the sets V_s and V_t ,

$$f(e) = \begin{cases} cap(e) & \text{if } e \in \langle V_s, V_t \rangle \\ 0 & \text{if } e \in \langle V_t, V_s \rangle \end{cases}$$

    Hence, *f* is a maximum flow, by Corollary 13.1.8.

Max-Flow Min-Cut Theorem

- **Theorem 13.2.4[Max-Flow Min-Cut]** For a given network , the value of a maximum flow is equal to the capacity of a minimum cut.
  - ✓ The *s*-*t* cut constructed in the proof of Theorem 13.2.3 has capacity equal to value of the maximum flow.
- The outline of an algorithm (with "ad hoc" implementation) for maximizing the flow in a network emerges from Proposition 13.2.1 and Theorem 13.2.3

Outline for Maximum Flow

Algo 13.2.1: Outline for Maximum Flow

Input: an s-t network N.
Output: a maximum flow f^* in network N.

[*Initialization*]
For each arc e in network N
  $f^*(e) := 0$
[*Flow Augmentation*]
While there exists an f^*-augmenting path in network N
  Find an f^*-augmenting path Q.
  Let $\Delta_Q = \min_{e \in Q}\{\Delta_e\}$.
  For each arc e of augmenting path Q
    If e is a forward arc
      $f^*(e) := f^*(e) + \Delta_Q$
    Else [e is a backward arc]
      $f^*(e) := f^*(e) - \Delta_Q$
Return flow f^*.

Finding an *f*-Augmenting Path

- **Review from 4.1** A frontier arc is an arc *e* directed from a labeled endpoint *v* to an unlabeled endpoint *w*.
- For constructing an *f*-augmenting path, frontier arc *e* is allowed to be backward (directed from vertex *w* to vertex *v*), and it can be added to the growing tree as long as it has slack $\Delta_e > 0$.
- **Terminology:** At any stage during tree-growing for constructing an *f*-augmenting path, let *e* be a frontier arc of tree *T*, with endpoints *v* and *w*. Then arc *e* is said to be usable if, for the current flow *f*, either $e \text{ runs } v \to w \text{ and } f(e) < cap(e)$, or $e \text{ runs } v \leftarrow w \text{ and } f(e) > 0.$
- **Example 13.2.4:** In Figure 13.2.7, if the choice of augmenting flow path at each iteration alternates between the directed path <*s*,*v*,*w*,*t*> and <*s*,*w*,*v*,*t*> , then the flow increases by only one unit at each iteration. Thus, it could take as many as *2M* iterations to obtain maximum flow.

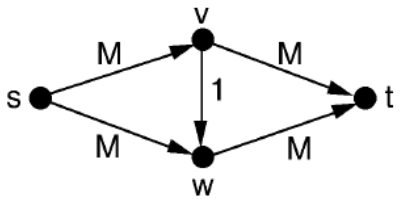


Figure 13.2.7 This network could require as many as $2M$ augmentations.

Finding an Augmenting Path

- Edmonds and Karp overcame such problems by employing *breadth-first search* to find an *f*-augmenting path **with the minimum number of arcs** or returns a minimum cut that indicates the current flow *f* is a maximum flow.

Algo 13.2.2: Finding an Augmenting Path

Input: a flow f in an s-t network N.
Output: an f-aug path Q,
or a mini s-t cut with capacity $val(f)$.

Initialize vertex set $V_s := \{s\}$.
Write label 0 on vertex s.
Initialize label counter $i := 1$
While vertex set V_s does not contain sink t
  If there are usable arcs
    Let e be a usable arc whose labeled endpoint v has the smallest possible label.
    Let w be the unlabeled endpoint of arc e.
    Set $backpoint(w) := v$.
    Write label i on vertex w.
    $V_s := V_s \cup \{w\}$
    $i := i + 1$
  Else
    Return s-t cut $\langle V_s, V_N - V_s \rangle$.
Reconstruct the f-aug path Q by following backpointers, starting from sink t.
Return f-augmenting path Q.

FFEK-Maximum Flow

- Algorithm 13.2.3 combine Algorithms 13.2.1 and 13.2.2. The "FFEK" in the algorithm name refers to Ford, Fulkerson, Edmonds, and Karp.

Algo 13.2.3: FFEK - Maximum Flow

Input: an s-t network N.
Output: a maximum flow f^* in network N.

[*Initialization*]
For each arc e in network N
    $f^*(e) := 0$
[*Flow Augmentation*]
Repeat
    Apply Algo 13.2.2 to find an f^*-augmenting path Q.
    Let $\Delta_Q = \min_{e \in Q}\{\Delta_e\}$.
    For each arc e of augmenting path Q
        If e is a forward arc **then**
            $f^*(e) := f^*(e) + \Delta_Q$
        else [e is a backward arc]
            $f^*(e) := f^*(e) - \Delta_Q$
until an f^*-aug path cannot be found in network N.
Return flow f^*.

FFEK-Maximum Flow

- **Example 13.2.5:** The following sequence of figures illustrates Algorithm 13.2.3 for the example network shown , starting with zero flow. The final figure shows an *s*-*t* cut with capacity equal to the value of the current flow , thereby establishing optimality.

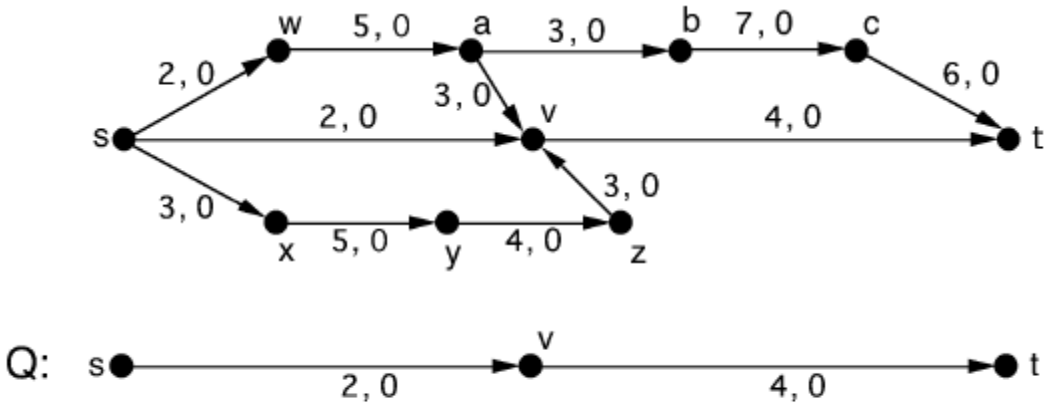


Figure 13.2.8 Iteration 0: $val(f) = 0$; augmenting path Q has $\Delta_Q = 2$.

FFEK-Maximum Flow

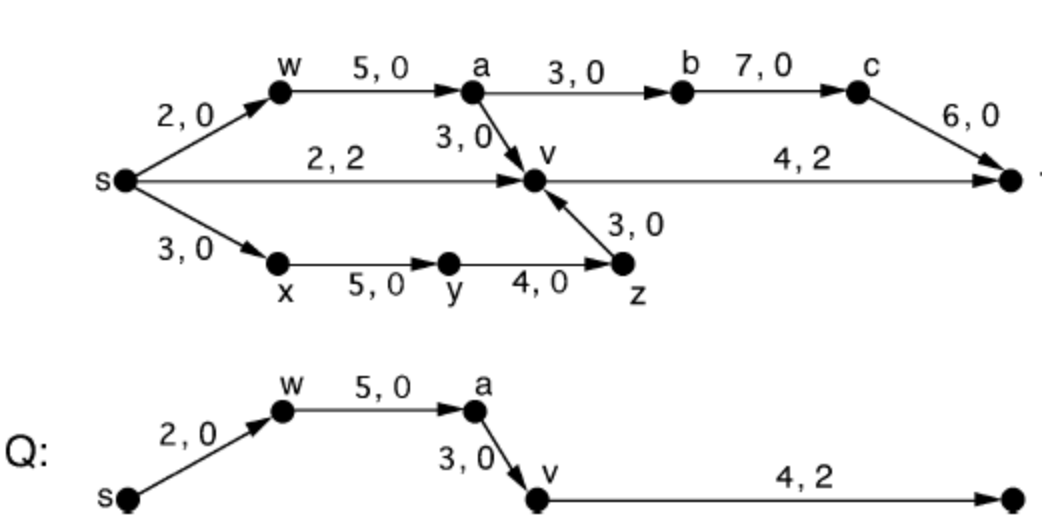


Figure 13.2.9 **Iteration 1:** $val(f) = 2$; augmenting path Q has $\Delta_Q = 2$.

FFEK-Maximum Flow

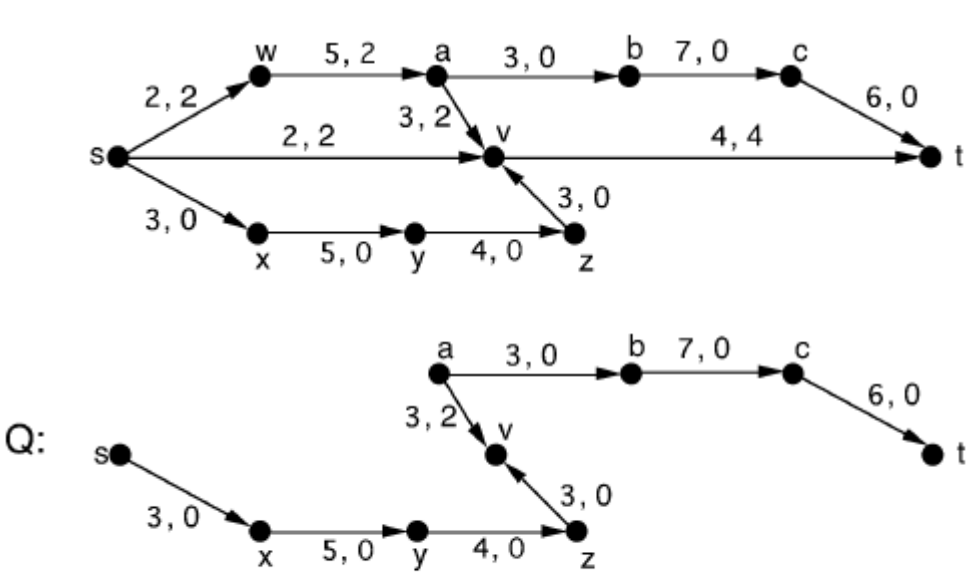
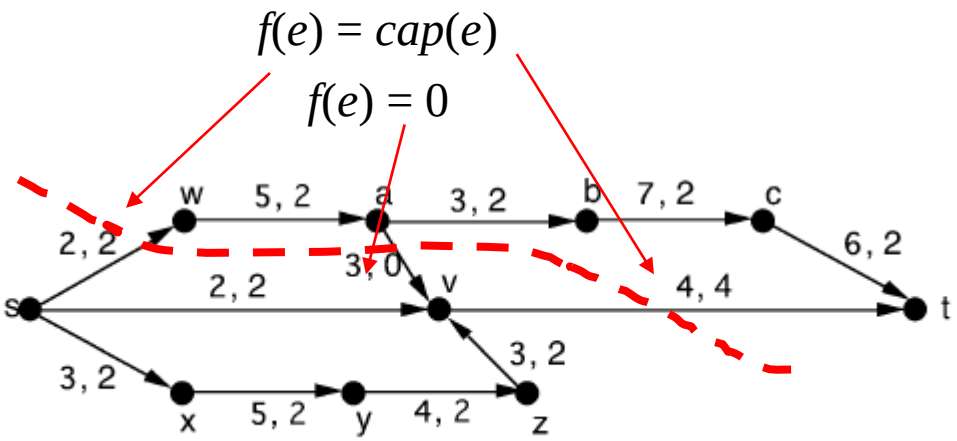


Figure 13.2.10 Iteration 2: $val(f) = 4$; augmenting path Q has $\Delta_Q = 2$.

FFEK-Maximum Flow



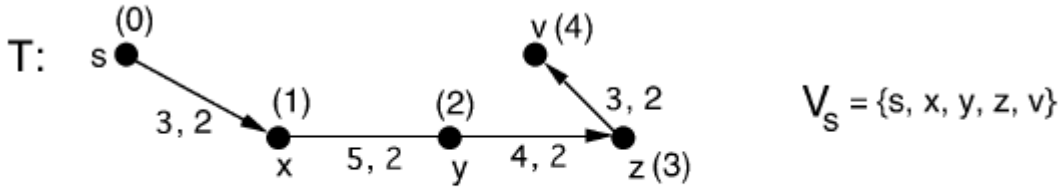


Figure 13.2.11 Final iteration:
$val(f) = 6 = cap\langle\{s, x, y, z, v\}, \{w, a, b, c, t\}\rangle$.

13.3 Flows and Connectivity

- **Lemma 13.3.1.** Let N be an s-t network such that $outdegree(s) > indegree(s)$, $indegree(t) > outdegree(t)$, and $outdegree(v) = indegree(v)$, for all other vertices v. Then there exist a directed s-t path in the network N.
  - ✓ Let W be a longest directed trail in N that starts at source s, and let z be its terminal vertex.
  - ✓ If vertex z were not the sink, then there would be an arc not in trail W that is directed from z (since $outdegree(z) = indegree(z)$). But this would contradict the maximality of trail W. Thus, W is a directed trail from source s to sink t.
  - ✓ If W has a repeated vertex, then part of W determines a directed cycle, which can be deleted from W to obtain a shorter directed s-t trail. This deletion step can be repeated until no repeated vertices remain, at which point, the resulting directed trail is an s-t path.

Flows and Connectivity

- **Proposition 13.3.2.** Let N be an s-t network such that

  $outdegree(s) - indegree(s) = m = indegree(t) - outdegree(t)$, and

  $outdegree(v) = indegree(v)$, for all other vertices $v \neq s, t$

  Then there exist m arc-disjoint directed s-t paths in N.

  - ✓ If m=1, then there exists an open eulerian directed trail T from source s to sink t, by Theorem 6.1.3. Theorem 1.5.2. trail T is either an s-t directed path or can be reduced to an s-t directed path.
  - ✓ By way of induction, assume that the assertion is true for $m = k$, for some $k \geq 1$, and consider a net work N for which the condition holds for $m = k+1$.
  - ✓ There exist a directed s-t path P by Lemma 13.3.1. If the arcs of P deleted from network N then the resulting network N satisfies the condition of the proposition for $m = k$.
  - ✓ By induction hypothesis, there exist k arc-disjoint directed s-t paths in network N. These k paths together with path P from a collection of $k+1$ arc-disjoint directed s-t paths in network N.

Theorem 6.1.3 A connected digraph has an open eulerian trail from vertex x to vertex y if and only if $indegree(x) + 1 = outdegree(x)$, $indegree(y) = outdegree(y) + 1$, and all vertices except x and y have equal indegree and outdegree.

Theorem 1.5.2. Let W be an open x-y walk. Then either W is an x-y path or there is an x-y path that is a reduced walk of W.

Two Basic Properties of 0-1 Networks

- **DEFINITION:** *A* **0-1 *network*** *is a capacitated network whose arc capacities are either* 0 *or* 1.
- **Proposition 13.3.3.** Let N be an s-t network such that $cap(e)=1$ for every arc e. Then the value of a maximum flow in network N equals the maximum number of arc-disjoint directed s-t paths in N.
  - ✓ Let f^* be a maximum flow in network N, and let r be the maximum number of arc-disjoint directed s-t paths in N. Consider the network N^*, obtained by deleting from N all arcs e such that $f^*(e) = 0$. Then $f^*(e) = 1$ for all arc e in N^*. It follows that the definitions that for every vertex v in network N^*,

$$\sum_{e\in Out(v)} f^*(e) = |Out(v)| = outdegree(v) \text{ and } \sum_{e\in In(v)} f^*(e) = |In(v)| = indegree(v)$$

  - ✓ Thus, by definition of $val(f^*)$ in N^* and by the conservation-of-flow property,

$$outdegree(s) - indegree(s) = val(f^*) = indegree(t) - outdegree(t),$$
$$\text{and } outdegree(v) = indegree(v), \text{ for all other vertices } v \neq s, t.$$

  - ✓ By proposition 13.3.2, there are $val(f^*)$ arc-disjoint directed s-t path in N^* and, hence, also in N, which implies $val(f^*) \leq r$.

Two Basic Properties of 0-1 Networks

- To obtain the reverse inequality, let $\{P_1, P_2, \ldots, P_r\}$ be a largest collection of arc-disjoint directed s-t paths in N, and consider the function $f : E_N \rightarrow R^+$ defined by

$$f(e) = \begin{cases} 1, & if\ some\ path\ P_i\ uses\ arc\ e \\ 0, & otherwise \end{cases}$$

- Then f is a feasible flow in network N, with $val(f) = r$. It follows that $val(f^*) \geq val(f) = r$.

Separating Sets and Cuts

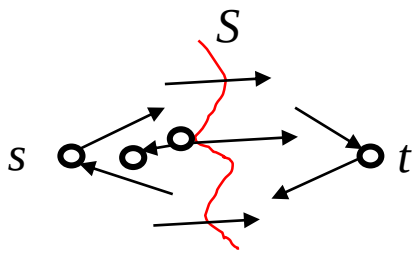
- **REVIEW §5.3**: : *Let s and t be distinct vertices in a connected graph G. A s-t* ***separating edge set*** *in G is a set of edges whose removal destroys all s-t paths in G.*
- **DEFINITION**: *Let s and t be distinct vertices in a digraph D. A s-t* ***separating arc set*** *in D is a set of arcs whose removal destroys all s-t paths in D.*
- **Proposition 13.3.4.** Let N be an s-t network such that $cap(e)=1$ for every arc e. Then the capacity of a minimum s-t cut in network N equals to minimum number of arcs in an s-t separating arc set in N.
  - ✓ Let K^* be a minimum s-t cut in network N, and let q be the minimum number of arcs in an s-t separating arc set in N. Since K^* is an s-t cut, it is also an s-t separating arc set. Thus, $cap(K^*) \geq q$.

Separating Sets and Cuts

- To obtain the reverse inequality, let S be an s-t separating arc set in network N, containing q arcs, and **let R be the set of all vertices in N that are reachable from source s by a directed path that contains no arc from set S.** Then, by the definitions of arc sets S and vertex set R, $t \notin R$, which means that $<R, V_N\text{-}R>$ is an s-t cut.
- Moreover, $<R, V_N\text{-}R> \subseteq S$. Therefore,

$$\begin{aligned} cap(K^*) &\leq cap< R, V_N\text{-}R > && \text{(since } K^* \text{ is a minimum } s\text{-}t \text{ cut)} \\ &= | < R, V_N\text{-}R > | && \text{(since all capacities are 1)} \\ &\leq | S | && \text{(since } < R, V_N\text{-}R > \subseteq S) \\ &= q \end{aligned}$$

which completes the proof.



Arc and Edge Versions of Menger's Theorem Revisited

- **Proposition 13.3.5.[Arc Form of Menger's Theorem].** Let *s* and *t* be distinct vertices in a digraph *D*. Then the maximum number of arc-disjoint directed *s*-*t* paths in *D* is equal to the minimum number of arcs in an *s*-*t* separating arc set of *D*.
  - ✓ Let *N* be the *s*-*t* network obtained by assigning a unit capacity to each arc of digraph *D*. Then the result follows from Propositions 13.3.3 and 13.3.4, together with the Max-Flow Min-Cut Theorem (Theorem 13.2.4).

13.3.3. Let *N* be an *s*-*t* network such that *cap*(*e*)=1 for every arc *e*. Then the value of a maximum flow in network *N* equals the **maximum number of arc-disjoint directed *s*-*t* path in *N***.
13.2.4. For a given network , the value of a maximum flow is equal to the capacity of a minimum cut.
13.3.4. Let *N* be an *s*-*t* network such that *cap*(*e*)=1 for every arc *e*. Then the capacity of a minimum *s*-*t* cut in network *N* equals to **minimum number of arcs in an *s*-*t* separating arc set in *N***.

- **Assertion 13.3.6.** Let *s* and *t* be distinct vertices in a graph *G*, and let $\overleftrightarrow{G}$ be the digraph obtained by replacing each edge *e* of *G* with a pair of oppositely directed arcs having the same endpoints as edge *e*. Then there is a one-to-one correspondence between the *s*-*t* paths in graph *G* and the directed *s*-*t* paths in digraph $\overleftrightarrow{G}$. Moreover, if two directed *s*-*t* paths in $\overleftrightarrow{G}$ are arc-disjoint, then their corresponding *s*-*t* paths in *G* are edge-disjoint.

Arc and Edge Versions of Menger's Theorem Revisited

- **Lemma 13.3.7.** Let S be a minimal s-t separating arc set of a digraph D. Then for every pair of vertices x, y in D, at most one of the arcs $a = xy$ and $b = yx$ is in S.
  - ✓ Suppose not. Let $P_a = \langle s, v_1, v_2, \ldots, v_j, x, y, v_{j+1}, v_{j+2}, \ldots, t \rangle$ be (the vertex sequence of) an s-t directed path in D whose only arc in S is $a = xy$,
  - ✓ and let $P_b = \langle s, w_1, w_2, \ldots, w_k, y, x, w_{k+1}, w_{k+2}, \ldots, t \rangle$ be an s-t directed path in D whose only arc in S is $b = yx$.
  - ✓ Then $P = \langle s, v_1, v_2, \ldots, v_j, x, w_{k+1}, w_{k+2}, \ldots, t \rangle$ is an s-t directed path in D that contains neither arc a nor arc b. Thus, P (and hence P_a and P_b) must contain some other arc in S, which contradicts the definition of path P_a and P_b.

Arc and Edge Versions of Menger's Theorem Revisited

- **Assertion 13.3.8.** Let s and t be distinct vertices of a graph G, and let $\overleftrightarrow{G}$ be the digraph obtained by replacing each edge e of G with a pair of oppositely directed arcs having the same endpoints as edge e. Then the minimum number of edges in an s-t separating edge set of graph G is equal to the minimum number of arcs in an s-t separating arc set of $\overleftrightarrow{G}$.
  - ✓ Let m be the size of a minimum s-t separating edge set S of G, and let $\overleftrightarrow{m}$ be the size of a minimum s-t separating arc set of $\overleftrightarrow{G}$. Let $\overleftrightarrow{S}$ be the arc set of $\overleftrightarrow{G}$ that results when each edge in S is replaced by its two oppositely directed arcs. Then $\overleftrightarrow{S}$ is an s-t separating arc set of $\overleftrightarrow{G}$ of size $2m$.
  - ✓ Let $\overleftrightarrow{S}^*$ be a minimal s-t separating arc set of $\overleftrightarrow{G}$ contained in $\overleftrightarrow{S}$. Lemma 13.3.7 implies that $|\overleftrightarrow{S}^*| \leq |S|$, and hence, $\overleftrightarrow{m} \leq m$.
  - ✓ Now let $\overleftrightarrow{T}^*$ be a minimum s-t separating arc set of $\overleftrightarrow{G}$. Then the corresponding edge set T that results from ignoring directions is an s-t separating edge set of G, from which it follows that $m \leq \overleftrightarrow{m}$.

Arc and Edge Versions of Menger's Theorem Revisited

- **Theorem 13.3.9[Edge Form of Menger's Theorem].** Let s and t be distinct vertices of a graph G. Then the maximum number of edge-disjoint s-t paths in G equals the minimum number of edges in an s-t separating edge set of graph G.

  - ✓ Let m and M be the size of a minimum s-t separating edge set and a maximum set of edge-disjoint s-t paths, respectively, in graph G, and let $\overleftrightarrow{m}$ and $\overleftrightarrow{M}$ be the sizes of a minimum s-t separating arc set and a maximum set of arc-disjoint s-t directed paths, respectively, in graph $\overleftrightarrow{G}$.

  - ✓ If $\overleftrightarrow{F}^*$ is a maximum set of arc-disjoint s-t directed paths in $\overleftrightarrow{G}$, then by Assertion 13.3.6, the corresponding set F of s-t paths in G is edge-disjoint, which implies

    $$\overleftrightarrow{M} = |\overleftrightarrow{F}^*| = |F| \leq M$$

  - ✓ By Assertion 13.3.8 and Theorem 13.3.5, we have

    $$\overleftrightarrow{m} = \overleftrightarrow{M} \leq M \leq m = \overleftrightarrow{m}$$

    where the second inequality follows because **it takes at least m edges to destroy *m* edge-disjoint paths**.

13.3.6. Then there is a one-to-one correspondence between the s-t paths in graph G and the directed s-t paths in digraph $\overleftrightarrow{G}$. Moreover, if two directed s-t paths in $\overleftrightarrow{G}$ are arc-disjoint, then their corresponding s-t paths in G are edge-disjoint.
13.3.5. Then the maximum number of arc-disjoint directed s-t paths in D is equal to the minimum number of arcs in an s-t separating arc set of D.
13.3.8. Then the minimum number of edges in an s-t separating edge set of graph G is equal to the minimum number of arcs in an s-t separating arc set of $\overleftrightarrow{G}$.

Determining Edge-Connectivity Using Network Flows

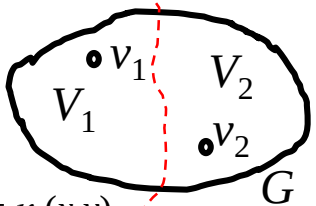
- **REVIEW FROM §5.1:** The edge-connectivity $\kappa_e(G)$ is the size of a smallest edge-cut in graph G.
- **DEFINITION:** The local edge-connectivity $\kappa_e(s,t)$ between distinct vertices s and t in a graph G is the minimum number of edges in an s-t separating edge set in G.
- **Proposition 13.3.10.** The edge-connectivity of a graph G is equal to the minimum of the local edge-connectivities, taken all pairs of vertices s an t. That is,

$$\kappa_e(G) = \min\{\kappa_e(s,t)\} \qquad \text{(where } s,t \in V_G\text{)}$$

Determining Edge-Connectivity Using Network Flows

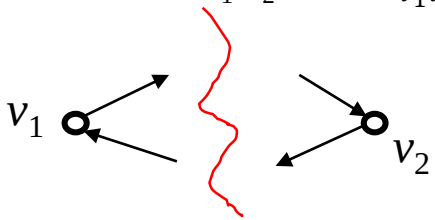
- **Proposition 13.3.11.** Let $<V_1, V_2>$ be a partition-cut of minimum cardinality in a graph G, and let v_1 and v_2 be any vertices in V_1 and V_2, respectively. Then the edge-connectivity $\kappa_e(G)$ equals the local edge-connectivity $\kappa_e(v_1,v_2)$.
  - ✓ Suppose that the minimum local edge-connectivity is achieved between vertices x and y. Then, $\kappa_e(G) = \kappa_e(x,y)$ (by Proposition 13.3.10). It suffice to show that $\kappa_e(v_1,v_2) \leq \kappa_e(x,y)$.
  - ✓ Let $\overleftrightarrow{G}$ be the digraph obtained by replacing each edge of graph G with two oppositely directed arcs. Then digraph $\overleftrightarrow{G}$ can be regarded as a v_1-v_2 capacitated network $\overleftrightarrow{G}_{v_1v_2}$ and as an x-y capacitated network $\overleftrightarrow{G}_{xy}$, where each arc is assigned unit capacity.
  - ✓ Let K^* be a minimum v_1-v_2 cut in network $\overleftrightarrow{G}_{v_1v_2}$. It follows that $cap(K^*) \leq cap<V_1,V_2>$, since the partition-cut $<V_1,V_2>$ corresponds to a v_1-v_2 cut in the network $\overleftrightarrow{G}_{v_1v_2}$.
  - ✓ Next, let f^* be a maximum flow and $<V_x, V_y>$ a minimum x-y cut in x-y network $\overleftrightarrow{G}_{xy}$, so that $cap<V_x, V_y> = val(f^*)$. Then the following chain inequalities establishes the desired inequality.

$<V_1, V_2>$ minimum partition cut



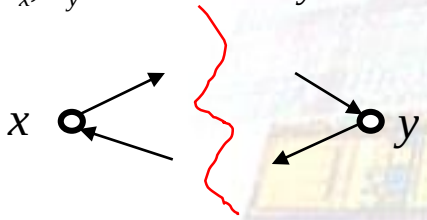
$\kappa_e(G) = \kappa_e(x,y)$

K^* minimum v_1-v_2 cut in $\overleftrightarrow{G}_{v_1v_2}$



f^* max flow;

$< V_x, V_y >$ minimum x-y cut in $\overleftrightarrow{G}_{xy}$

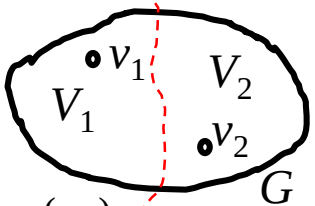


Determining Edge-Connectivity Using Network Flows

$$
\begin{aligned}
\kappa_e(v_1,v_2) &= cap(K^*) && \text{(Proposition 13.3.4 and Assertion 13.3.8)}\\
&\le cap\langle V_1,V_2\rangle && (\langle V_1, V_2\rangle \text{ corresponds to a } v_1\text{-}v_2 \text{ cut in } \overleftrightarrow{G}_{v_1v_2})\\
&= |\langle V_1,V_2\rangle| && \text{(all arcs have unit capacity)}\\
&\le |\langle V_x,V_y\rangle| && (\langle V_x, V_y\rangle \text{ corresponds to a partition-cut in } G)\\
&= cap\langle V_x,V_y\rangle && \text{(all arcs have unit capacity)}\\
&= val(f^*) && \text{(Max-Flow Min-Cut Theorem 13.2.4)}\\
&= \kappa_e(x,y) && \text{(Proposition 13.3.3 and Theorem 13.3.5)}
\end{aligned}
$$

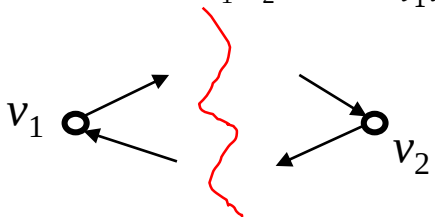
13.3.8. Then the minimum number of edges in an *s*-*t* separating edge set of graph *G* is equal to the **minimum number of arcs in an *s*-*t* separating arc set of** $\overleftrightarrow{G}$.
13.3.4. Let *N* be an *s*-*t* network such that *cap*(*e*)=1 for every arc *e*. Then the capacity of a minimum *s*-*t* cut in network *N* equals **to minimum number of arcs in an *s*-*t* separating arc set in *N***.
13.3.3. Let *N* be an *s*-*t* network such that *cap*(*e*)=1 for every arc *e*. Then the value of a maximum flow in network *N* equals **the maximum number of arc-disjoint directed *s*-*t* path in *N***.
13.3.5. Then **the maximum number of arc-disjoint directed *s*-*t* paths in *D*** is equal to the minimum number of arcs in an *s*-*t* separating arc set of *D*.

$\langle V_1, V_2\rangle$ minimum partition cut

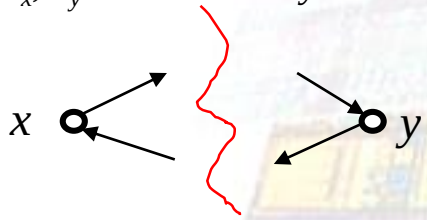


$\kappa_e(G) = \kappa_e(x,y)$.

K^* minimum v_1-v_2 cut in $\overleftrightarrow{G}_{v_1v_2}$



f^* max flow;
$\langle V_x, V_y\rangle$ minimum x-y cut in $\overleftrightarrow{G}_{xy}$



Determining Edge-Connectivity Using Network Flows

- **Corollary 13.3.12.** Let s be any vertex in graph G. Then
$$\kappa_e(G) = \min\{\kappa_e(s,t)\} \qquad (\text{where } t \in V_G\text{-}\{s\})$$
  - ✓ Let $< V_1, V_2 >$ be a partition-cut of minimum cardinality, and suppose, without loss of generality, that vertex $s \in V_1$. There must be some vertex $t \in V_2$ (otherwise, $E_G = \emptyset$, and the assertion would be trivially true). By Proposition 13.3.11, it follows that $\kappa_e(G) = \kappa_e(s,t)$.
- The variable κ_e, used in the algorithm, represents the edge-connectivity of graph G and is initialized with the sufficiently positive integer $|E_G|$.

Algorithm 13.3.1: Edge-Connectivity Calculation
Input: a graph G
Output: the edge-connectivity κ_e of graph G
Construct digraph $\overleftrightarrow{G}$
Let s be an arbitrary vertex of graph G
Initialized edge-connectivity $\kappa_e := |E_G|$
For each vertex $t \in V_G\text{-}\{s\}$
Apply Algorithm 13.2.3 to s-t network $\overleftrightarrow{G}$ to obtain maximum flow f^*,
If $val(f^*) < \kappa_e$
$\kappa_e := val(f^*)$
Return κ_e

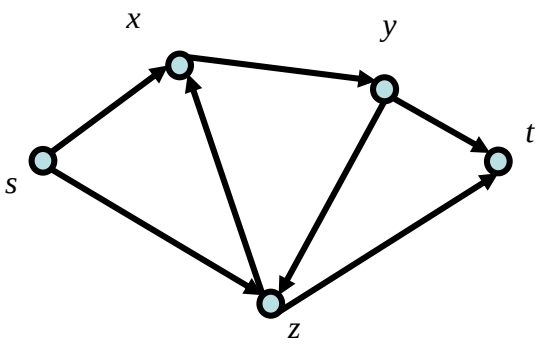
Determining Edge-Connectivity Using Network Flows

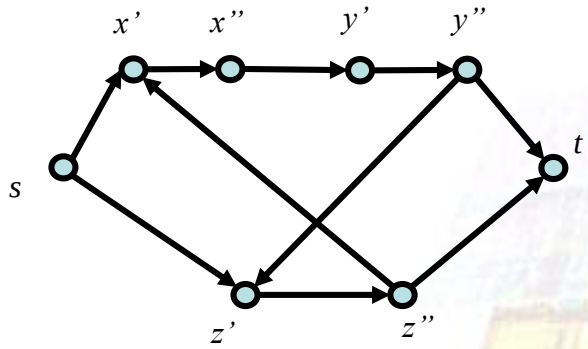
- **COMPUTATIONAL NOTE:** Algorithm 13.3.1 reqeires $O(n)$ iterations, and since Algorithm 13.2.3 requires $O(n|E|^2)$ computations, the overall computation of Algorithm 13.3.1 is $O(n^2|E|^2)$. This can be reduced if one of the more efficient maximum-flow algorithms cited earlier is used instead of Algorithm 13.2.3.

Using Network Flows to Prove the Vertex Forms of Menger's Theorem

Construction of Digraph N^D From a Digraph D:

- Let s and t be any pair of non-adjacent vertices in a digraph D. The digraph N^D is obtained from digraph D as follows:
  - Each vertex $x \in V_D\text{-}\{s, t\}$ corresponds to two vertices x' and x'' in digraph N^D and an arc directed from x' to x'' .
  - Each arc in digraph D that is directed from vertex s to a vertex $x \in V_D\text{-}\{s, t\}$ corresponds to an arc in digraph N^D directed from s to x'.
  - Each arc in D that is directed from a vertex $x \in V_D\text{-}\{s, t\}$ to vertex t corresponds to an arc in N^D directed from x'' to t.
  - Each arc in D that is directed from a vertex $x \in V_D\text{-}\{s, t\}$ to vertex $y \in V_D\text{-}\{s, t\}$ corresponds to an arc in N^D directed from x'' to y'.





Using Network Flows to Prove the Vertex Forms of Menger's Theorem

- ✓ **REVIEW FROM §5.3:** Let s and t be a pair of non-adjacent vertices in a graph G (or digraph D). An s-t separating vertex set in G (or D) is a set of vertices whose removal destroys all s-t paths in G (or all directed paths in D).
- ✓ Thus, an s-t separating vertex set is a set of vertices that contains at least one internal vertex of every (directed) s-t path.

- **DEFINITION:** Two (directed) s-t paths in a digraph are internally disjoint if they have no internal vertices in common.

Relationships Between Digraph D and N^D

- **Assertion 13.3.13.** *There is a one-to-one correspondence between directed s-t paths in digraph D and directed s-t paths in digraph N^D.*
- **Assertion 13.3.14.** *Two directed s-t paths in D are internally disjoint if and only if their corresponding s-t directed paths in N^D are arc-disjoint.*
- **Assertion 13.3.15.** *The maximum number of internally disjoint directed s-t paths in D is equal to the maximum number of arc-disjoint directed s-t paths in N^D.*
- **Assertion 13.3.16.** *The minimum number of vertices in an s-t separating vertex set in digraph D is equal to the minimum number of arcs in an s-t separating arc set in digraph N^D.*
- **Theorem 13.3.17 [Vertex Form of Menger for Digraphs].** *Let s and t be a pair of non-adjacent vertices in a digraph D. Then the maximum number of internally disjoint directed s-t paths in D is equal to the minimum number of vertices in an s-t separating vertex set in digraph D.*
  - ✓ This follows from Assertions 13.3.13 through 13.3.16, together with the arc form of Menger's Theorem (Theorem 13.3.5).

Vertex Form of Menger for Undirected Graphs

- **Theorem 13.3.18 [Vertex Form of Menger for Undirected Graphs].** Let *s* and *t* be a pair of non-adjacent vertices in a graph *G*. Then the maximum number of internally disjoint *s*-*t* paths in *G* is equal to the minimum number of vertices in an *s*-*t* separating vertex set in *G*.
  - ✓ Let $\overleftrightarrow{G}$ be the digraph obtained by replacing edge *e* of *G* with a pair of oppositely directed arcs having the same endpoints as edge *e*. The result follows by Theorem 13.3.17 and Assertions 13.3.6 and 13.3.8.

13.3.17. *Let s and t be a pair of non-adjacent vertices in a digraph D.* ***Then the maximum number of internally disjoint directed s-t paths in D is equal to the minimum number of vertices in an s-t separating vertex set in digraph D.***

13.3.6. Then there is **a one-to-one correspondence between the *s*-*t* paths in graph *G* and the directed *s*-*t* paths in digraph $\overleftrightarrow{G}$.** Moreover, if two directed *s*-*t* paths in $\overleftrightarrow{G}$ are arc-disjoint, then their corresponding *s*-*t* paths in *G* are edge-disjoint.

13.3.8. Let *s* and *t* be distinct vertices of a graph *G*, and let $\overleftrightarrow{G}$ be the digraph obtained by replacing each edge *e* of *G* with a pair of oppositely directed arcs having the same endpoints as edge *e*. Then **the minimum number of edges in an *s*-*t* separating edge set of graph *G* is equal to the minimum number of arcs in an *s*-*t* separating arc set of $\overleftrightarrow{G}$.**

Determining Vertex-Connectivity Using Network Flows

- **Remark**: The following algorithm, which has $O(n|E_G{}^3|)$ time-complexity, computes the vertex-connectivity of an n-vertex graph by calculating the local vertex-connectivity between various pairs of non-adjacent vertices.

Algorithm 12.3.2: Vertex-Connectivity Calculation

Input: an graph G with $V_G = \{v_1, v_2, \ldots v_n\}$.
Output: the vertex-connectivity κ_v of graph G.

Construct digraph $\overleftrightarrow{G}$.
Initialize vertex-connectivity $\kappa_v := |V_G|$.
Initialize index $k := 0$.
While $k \leq \kappa_v$
  $k := k + 1$
  For $j = k + 1$ to n
    If vertices v_k and v_j are not adjacent
      Construct digraph $N^{\overleftrightarrow{G}}$.
      Assign unit capacity to each arc in digraph $N^{\overleftrightarrow{G}}$.
      Apply Algorithm 12.2.3 to network $N^{\overleftrightarrow{G}}$ with source v_k and sink v_j to obtain maximum flow f^*.
      If $val(f^*) < \kappa_v$
        $\kappa_v := val(f^*)$
Return κ_v.

13.4 Matchings, Transversals, and Vertex Covers

- **TERMINOLOGY**: If e, with endpoints x and y, is an edge in a matching M, then M ***matches vertex*** x ***with vertex*** y, or x ***is matched with*** y ***by*** M.
- **DEFINITION**: A ***maximum (-cardinality) matching*** is a matching with the greatest number of edges.
- **Remark**: A *maximal* matching in a graph G is a matching that is not a proper subset of any other matching in G. Clearly, every maximum matching is a maximal matching, but a maximal matching need not to be a maximum one.
- **Example 13.4.1.** Consider the graph G shown in Figure 13.4.1. The singleton edge sets $\{b\}$, $\{c\}$, $\{d\}$ and $\{e\}$ are the nonempty matchings in graph G that are *not* maximal. The edge set $\{a\}$ is a maximal matching in G that is not maximum, and the edge sets $\{b, d\}$, $\{c, e\}$ are the maximum matchings in G.

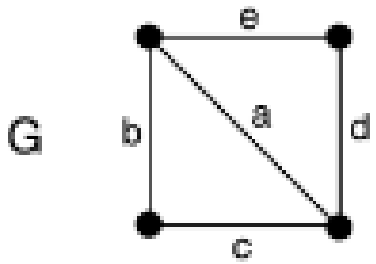


Figure 13.4.1 Two of the three maximal matchings in G are maximum.

Maximum Matching in a Bipartite Graph

- **Application 13.4.1** *The Personnel-Assignment Problem*: Suppose that a company requires a number of different types of jobs, and suppose each worker is suited for some of these jobs, but not others. Assuming that each person can perform at most one job at a time, how should the jobs be assigned so that the maximum number of jobs can be performed simultaneously?
- The bipartite graph of Figure 13.4.2 has vertex bipartition $\{V_J, V_W\}$, where the solid vertices represent the jobs, and the hollow vertices represent the workers. Each edge corresponds to a suitable job assignment. The maximum matching whose edges appear in bold shows that all five jobs can be assigned to workers.

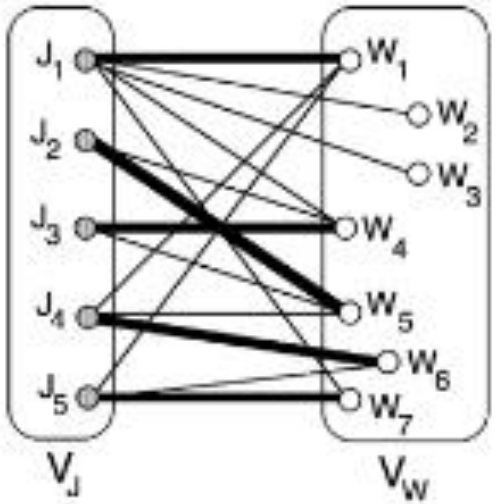


Figure 13.4.2 A maximum matching that solves the assignment problem.

Converting a Maximum-Matching Problem into a Maximum-Flow Problem

- Let G be a bipartite graph with vertex bipartite $\{X, Y\}$. Then the problem of finding a maximum matching in graph G can be transformed into a maximum-flow problem in an s-t network $\vec{G}_{st}$, constructed from graph G as follows:
  - ✓ (1) The vertex-set of network $\vec{G}_{st}$ is $V_G \cup \{s, t\}$, where s and t are two new vertices that are the source and sink, respectively, of $\vec{G}_{st}$.
  - ✓ (2) Each edge in graph G between a vertex $x \in X$ and a vertex $y \in Y$ corresponds to an arc in network $\vec{G}_{st}$ directed from vertex x to vertex y.
  - ✓ (3) For each vertex $x \in X$, an arc in network $\vec{G}_{st}$ is drawn from source s to vertex x.
  - ✓ (4) For each vertex $y \in Y$, an arc in network $\vec{G}_{st}$ is drawn from vertex y to sink t.
  - ✓ (5) Each arc e in network $\vec{G}_{st}$ is assigned a capacity $cap(e) = 1$.
- **Example 13.4.2** Figure 13.4.3 shows the bipartite graph from Application 13.4.1 and the corresponding s-t network $\vec{G}_{st}$.

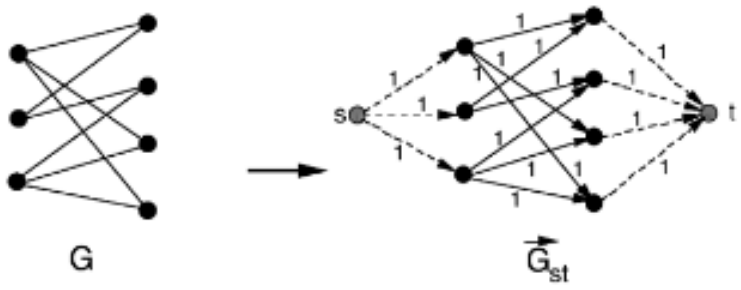


Figure 13.4.3 A bipartite graph G and its corresponding network $\vec{G}_{st}$.

Relationship Between Matchings in G and Flows in $\overrightarrow{G}_{st}$

- **Proposition 13.4.1.** *Let G be a bipartite graph and $\vec{G}_{st}$ the s-t network constructed from G. Then there is a one-to-one correspondence between the integral flows of network $\vec{G}_{st}$ and the matching in graph G.*

  - ✓ Let f be an integral-valued flow in the network $\vec{G}_{st}$. Then the unit capacities imply that $f(e) = 0$ or 1, for each arc e in network $\vec{G}_{st}$. Let M be the set of edges in graph G whose corresponding arcs in network $\vec{G}_{st}$ have unit flow. It follows from the structure of $\vec{G}_{st}$ and the conservation-of-flow property that edge set M is a matching in graph G.
  - ✓ Conversely, let M be a matching in graph G, and for any edge e in graph G, let $\vec{e}$ denote the corresponding arc in network $\vec{G}_{st}$. Then for every arc $a \in \vec{G}_{st}$, the function defined by

$$f(a) = \begin{cases} 1, & \text{if } a = \vec{e}, \text{ for some edge } e \in M \\ 1, & \text{if arc } a \text{ is adjacent to } \vec{e}, \text{ for some edge } e \in M \\ 0, & \text{otherwise} \end{cases}$$

  is a flow in network $\vec{G}_{st}$.

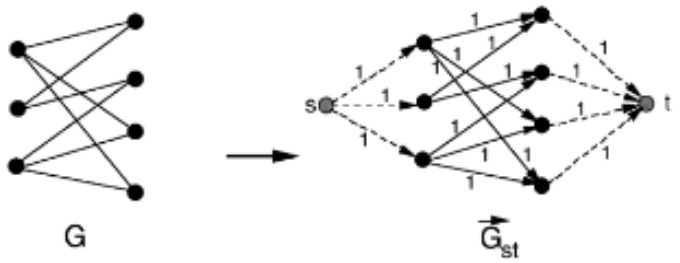
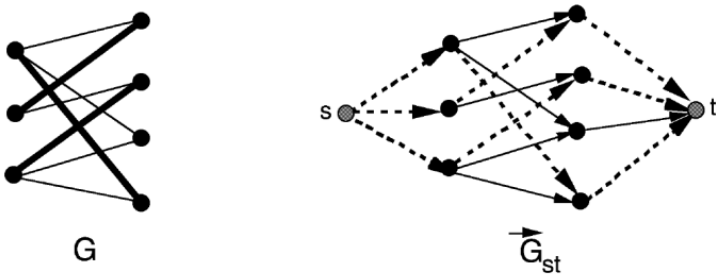


Figure 13.4.3 A bipartite graph G and its corresponding network $\vec{G}_{st}$.

Relationship Between Matchings in G and Flows in $\overrightarrow{G}_{st}$

- **Corollary 13.4.2.** *Let M be a matching in bipartite graph G and f be the corresponding flow in network* $\vec{G}_{st}$. *Then val(f) = |M|, and f is a maximum flow if and only if M is a maximum matching.*
  - ✓ This follows from the correspondence established in Proposition 13.4.1.
- **Example 1.4.3.** On the left in Figure 13.4.4 below is a maximum matching of size 3 in a bipartite graph *G*. Its corresponding maximum flow un network $\vec{G}_{st}$ has unit flow in each arc represented as a dashed line.
  - ✓ Algorithm 13.4.1 finds a maximum matching in a bipartite graph *G* by finding a maximum flow in the corresponding network $\vec{G}_{st}$, as prescribed by Proposition 13.4.1.



Algorithm 13.4.1: Maximum Bipartite Matching

Input: a bipartite graph G with vertex bipartition $\{X, Y\}$.
Output: a maximum matching M of graph G.

Initialize edge set $M := \emptyset$.
Construct the s-t network $\vec{G}_{st}$ that corresponds to bipartite graph G.
Apply Algorithm 13.2.3 to network $\vec{G}_{st}$ to obtain maximum flow f^*.
For each arc $\vec{e}$ in network $\vec{G}_{st}$
    If $f^*(\vec{e}) = 1$
        $M := M \cup \{e\}$
Return edge set M.

Relationship Between Matchings in G and Flows in $\overrightarrow{G}_{st}$

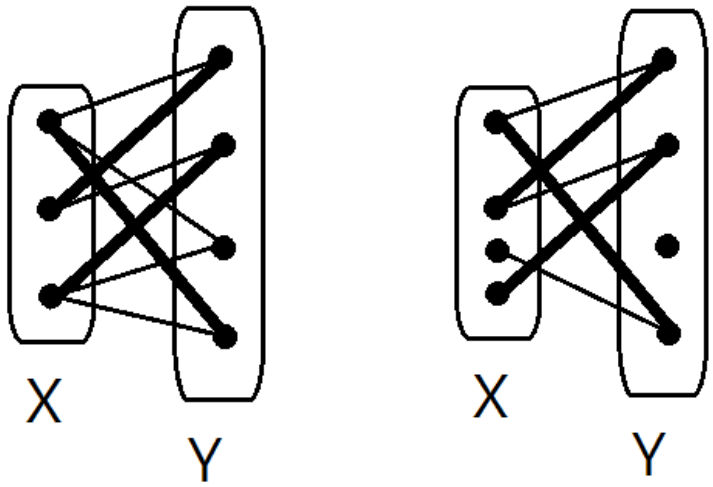
- **Remark:** One can view the algorithm above as iteratively augmenting a matching M by finding an M-augmenting path, analogous to the flow-augmenting paths of the maximum-flow algorithm. Edmonds [Ed65b] extended this notion by designing an algorithm for finding a maximum matching in a general graph. The Edmonds algorithm is more complicated than a bipartite-matching algorithm, but it and its more recent improvements have running time not much beyond that needed for bipartite matching. The following application illustrates the need for a more general algorithm.
- **Application 13.4.2.** *Pair Volunteers for Rescue Mission:* Suppose that several first-aid workers from around the world have volunteered for a rescue mission in some country that has been struck by a disaster. The volunteers are to be divided into two-person teams, but the members of each team must speak the same language. What is the maximum number of pairs that can be sent out on a rescue mission?
- A graph model for this problem has a vertex corresponding to each volunteer, and an edge exists between a pair of vertices if the corresponding volunteers speak the same language. Then a maximum matching in the graph corresponds to a maximum set of two-person teams.

Transversals for Families of Subsets

- **DEFINITION**: Let A be a nonempty finite set, and let $F = \{S_1, S_2, \ldots, S_r\}$ be a family of (not necessarily distinct) nonempty subsets of set A. Then a transversal (or system of distinct representatives) of family F is a sequence $T = \langle a_1, a_2, \ldots, a_r \rangle$ of r distinct elements of set A, such that $a_i \in S_i$, for each $i = 1, \ldots, r$.
- **Example 13.4.4.** Let $A = \{a, b, c, d, e\}$, and suppose that $F = \{S_1, S_2, \ldots, S_5\}$, where $S_1 = \{a, b\}$; $S_2 = \{b, c, d\}$; $S_3 = \{c, d, e\}$; $S_4 = \{d, e\}$; and $S_5 = \{e, a, b\}$ Then $T = \langle b, c, e, d, a \rangle$ is a transversal for family F.
- **Application 13.4.3.** *Pair Interns with Hospitals:* Suppose that r medical school graduates have applied for internships at various hospitals. For the ith medical school graduate, let S_i be the set of all hospitals that find applicant i acceptable. Then a transversal for a family $F = \{S_1, S_2, \ldots, S_r\}$ would assign each potential intern to a hospital that is willing to take him or her, such that no hospital gets assigned more than one intern.

Relationship Between Bipartite Matchings and Transversals

- **DEFINITION**: Let G be a bipartite graph with vertex bipartition $\{X, Y\}$, and let edge set M be a matching in G. Then the matching M is said to be ***X-saturating*** if each vertex in the set X is an endpoint of an edge in M, and to be ***Y-saturating*** if each vertex in the set Y is an endpoint of an edge in M.
- **Example 13.4.5.** The maximum matching shown on the left in Figure 13.4.5 is X-saturating, but the one on the right is not.



- **TERMINOLOGY NOTE**: Because a matching matches vertices in the X-set to vertices in the Y-set, some authors refer to an X-saturating matching as a *complete matching from X to Y*.

Finding a Transversal by Finding an X– Saturating Matching

- Let $A = \{ a_1 , a_2 , \dots , a_n \}$ be a set of n elements and $F = \{ S_1 , S_2 , \dots , S_r \}$ be a family of (not necessarily distinct) subsets of set A. Consider the following bipartite graph G constructed from F.
- The vertex bipartition $\{ X , Y \}$ of graph G consists of two vertex sets

$$X = \{ x_1 , x_2 , \dots , x_r \} \text{ and } Y = \{ y_1 , y_2 , \dots , y_n \}$$

Where each vertex $x_i \in X$ corresponds to the subset S_i in family F, and each vertex $y_j \in Y$ corresponds to the element $a_j \in A$. An edge exists between vertex x_i and vertex y_j if $a_j \in S_i$.

Then a transversal for family F corresponds to an X–saturating matching of graph G .

- **Example 13.4.6:** Figure 13.4.6 shows the X–saturating matching corresponds to the transversal $T = < b, c, e, d, a >$ from Example 13.4.4.

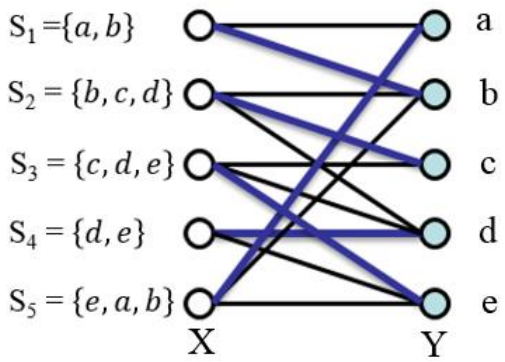


Figure 13.4.6 Transversal $T = < b, c, e, d, a >$ and corresponding matching.

Hall's Theorem

- The next theorem, known as *Hall's Theorem,* gives a necessary and sufficient condition for a bipartite graph to have an X–saturating matching. Its corollary, which is simply a restatement in terms of a family of subsets, characterizes those families of subsets that have a transversal. The graph version uses the following notation.
- **NOTATION:** Let G be a bipartite graph with vertex bipartite $\{X,Y\}$,and let W be a subset of vertex set X . Then $N(W)$ denotes the subset of Y consisting of all vertices in Y that are adjacent to at least one vertex in set W. In other words, $N(W)$ is the set of *neighbors* of set W.
- **DEFINITION:** A bipartite graph with vertex bipartition $\{X,Y\}$ is said to satisfy *Hall's Condition* if $|W| \leq |N(W)|$ for every subset W of X .

Hall's Theorem

- **Theorem 13.4.3 [Hall's Theorem for Bipartite Graphs, 1935] :** Let G be a bipartite graph with vertex bipartite $\{X,Y\}$. Then graph G has an X–saturating matching if and only if for each subset W of X , $|W| \leq |N(W)|$.

  - ✓ *Necessity* (→) Suppose that there exists an X–saturating matching M in graph G , and let W be any subset of X. Then every vertex $w \in W$ is matched to a vertex $y \in Y$ by an edge in matching M . Each such y is in the neighbor set $N(W)$ of W, and, thus , $|W| \leq |N(W)|$.

  - ✓ *Sufficiency* (←) Suppose that Hall's Condition holds, that is, $|W| \leq |N(W)|$, for every subset W of X . Consider the s-t network $\vec{G}_{st}$ corresponding to the bipartite graph G (as in Proposition 13.4.1). An X-saturating matching in graph G would be a maximum matching in G and would correspond to a maximum flow in network $\vec{G}_{st}$ with value $|X|$. Thus, by the Max-Flow Min-Cut Theorem (Theorem 13.2.4), it suffices to show that a minimum s-t cut in network $\vec{G}_{st}$ has capacity $|X|$. The s-t cut $<\{s\} , X \vee Y \vee \{t\} >$ has capacity $|X|$, so it remains to show that every other s-t cut K has capacity $cap(K) \geq |X|$.

✓

Hall's Theorem

Let $<V_s,V_t>$ be any s-t cut in network $\vec{G}_{st}$, and let $W = V_s \wedge X$. Then the cut $<V_s,V_t>$ can be expressed as the disjoint union of three arc sets (depicted as dashed lines in Figure 13.4.7), that is

$$< V_s , V_t > = < \{s\}, V_t \wedge X > \vee <W , V_t \wedge Y> \vee < V_s \wedge Y , \{t\} >$$

where $<A,B>$ denotes the set of arcs directed from vertex in A to a vertex in B .

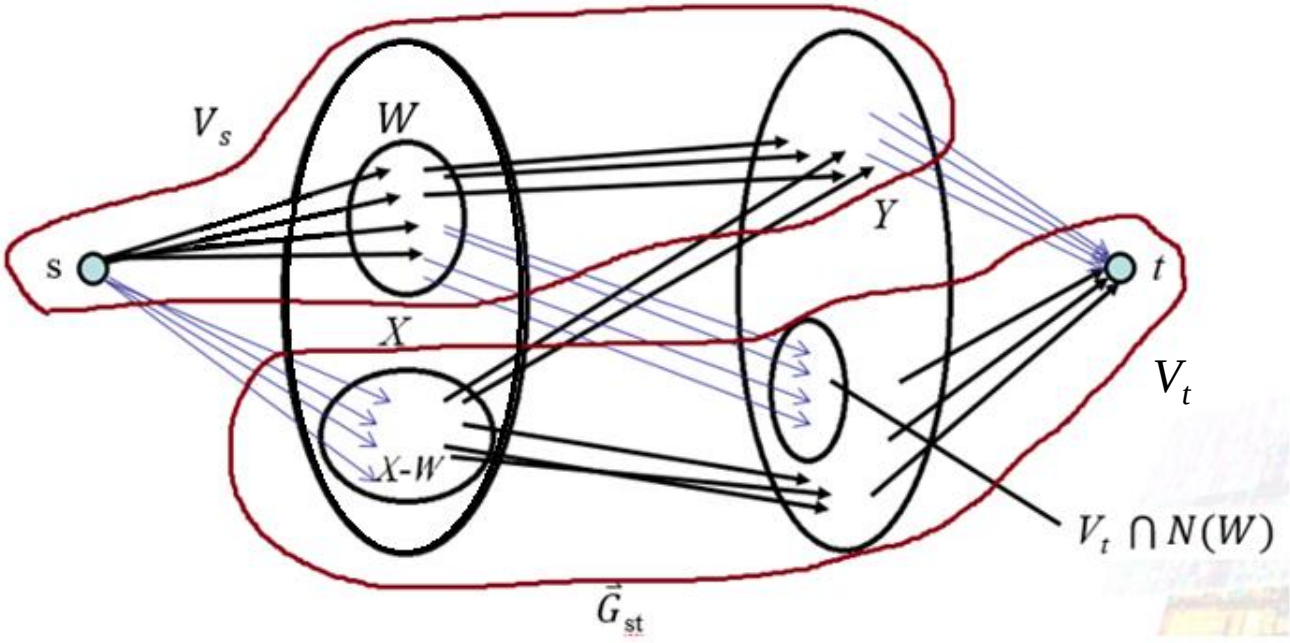


Figure 13.4.7 $< V_s , V_t > = <\{s\}, X\text{-}W > \vee <W, V_t \wedge Y> \vee < V_s \wedge Y, \{t\}>$

Hall's Theorem

The following chain of inequalities completes the proof.

$$
\begin{aligned}
cap\langle V_s, V_t\rangle &= |\langle\{s\}, X - W\rangle| + |\langle W, V_t \cap Y\rangle| + |\langle V_s \cap Y, \{t\}\rangle| \quad \text{(all capacites = 1)} \\
&= |X - W| + |\langle W, V_t \cap Y\rangle| + |V_s \cap Y| \quad \text{(by the construction of } \vec{G}_{st}\text{)} \\
&\geq |X - W| + |V_t \cap N(W)| + |V_s \cap Y| \quad \text{(by definition of } N(W)\text{)} \\
&= |X - W| + |N(W)| - |V_s \cap N(W)| + |V_s \cap Y| \quad \text{(see Figure 13.4.7)} \\
&\geq |X - W| + |N(W)| - |V_s \cap Y| + |V_s \cap Y| \quad \text{(since } N(W) \subseteq Y\text{)} \\
&= |X - W| + |N(W)| \\
&\geq |X - W| + |W| \quad \text{(by Hall's Condition)} \\
&= |X|
\end{aligned}
$$

Hall's Theorem

- **Corollary 13.4.4 [Hall's Theorem for Transversals]** Let A be a nonempty finite set, and $F = \{ S_1, S_2, \ldots, S_n \}$ be a family of nonempty subsets of set A. Then family F has a transversal if and only if the union of any k of the subset S_i contains at least k elements of set A $(1 \leq k \leq r)$
  - ✓ This is Hall's Theorem for Bipartite Graphs, restated in terms of transversals.
- **Remark:** One of the earliest incarnations of Hall's Theorem appeared as a solution to the following problem, known as the *Marriage Problem*: *Given a set of women, each of whom knows a subset of men, under what conditions can each of the women marry a man whom she knows ?* (See Exercise.)

Two Graph Factorization Theorems

- **Review from 9.4:**
  - A **factor** of a graph is a spanning subgraph.
  - A **factorization** of a graph G is a set of factors whose edge-sets from a partition of the edge-set E_G.
  - A k-**factor** of a graph is a k-regular factor of G.
  - A k-**factorization** of a graph is a factorization into k-factors.
- **Theorem 13.4.5 [König's 1-Gactorization Theorem]** [Kö16] Every r-regular bipartite graph G with $r > 0$ is 1-fractorable.
  - ✓ Suppose that X and Y are the two partites sets, and $W \subseteq X$. The number of edges from W to Y is $r|W|$. Since each vertex of Y has degree r, at most r of these edges are incident on any one vertex of Y, from which it follows (by the generalized pigehole principle) that

$$N(W) \geq \left\lceil \frac{r|W|}{r} \right\rceil = |W|$$

  - ✓ By Hall's Theorem for Bipartite Graphs, the graph G has an X-saturating matching. Since $|X| = |Y|$, such a matching is a 1-factor in G. The graph obtained from G by deleting the edges of this 1-factor is an $(r-1)$-regular bipartite graph, and the result follow by induction.

Two Graph Factorization Theorems

- **Review from §1.5 and §4.5 :**
  - An **eulerian trail** in a graph is a trail that contains every edge of that graph.
  - An **eulerian tour** is a closed eulerian trail.
  - An **eulerian graph** is a graph that has an eulerian tour.
  - **Eulerian-Graph Characterization**: A connected graph G is eulerian if and only if the degree of every vertex in G is even.
- **Theorem 13.4.6 [Petersen's 2-Factorization Theorem]** [Pe1891] Every regular graph G of even degree is 2-factorable.
  - ✓ We may assume that G is connected since a graph is factorable if and only if each of its components is factorable. Let G be a $2r$-regular graph with vertices $v_1, \ldots, v_n$, and let C be a closed eulerian trail in G, whose existence is guaranteed by the Eulerian-Graph Characterization, cited above. We define a new bipartite graph H with partite sets

$$U = \{u_1, \ldots, un\} \text{ and } W = \{w_1, \ldots, w_n\}$$

  - ✓ such that u_i and w_j are adjacent if vertex v_j immediately follows vertex v_i on the eulerian trail C. Graph H is r-regular, because the trail C enters and leaves each vertex of G exactly r times.

Two Graph Factorization Theorems

- Since H is a regular bipartite graph, it follows from Theorem 13.4.5 that H has a 1-factor F . By the construction of H, the edge $e \in F$ with endpoint u_i and w_j corresponds to edge e' if G with endpoints v_i and v_j. Moreover, for each subscript $k = 1,\ldots,n$, the vertices u_k and w_k both occur exactly once in the 1-factor F, and hence, the edge set

$$F' = \{e' | e \in F\}$$

- contains exactly two edges of g that are incident on the vertex v_k .It follows F is a 2-factor of G. The graph obtained from G by deleting the edges of this 2-factor is a $(2r-2)$-regular graph, and the result follows by induction.

Maximum Matchings and Minimum Vertex Covers

- The theme of *max-min* pairs of optimization problems, seen earlier in this chapter and in Chapter 5, appears once again in the context of *vertex cover* .
- **DEFINTION:** Let *G* be a graph, and *C* be a subset of the vertices of *G*. Then set *C* is a *vertex cover* of graph *G* if and only if every edge of *G* is incident on at least one vertex in *C*.
- **DEFINTION:** A *minimum vertex cover* is a vertex cover with the least number of vertices .
- **Example 13.4.7:** For the bipartite graph shown in Figure 13.4.8, a maximum matching (the solid vertices) and minimum vertex cover (the bold vertices) both have cardinality 5.

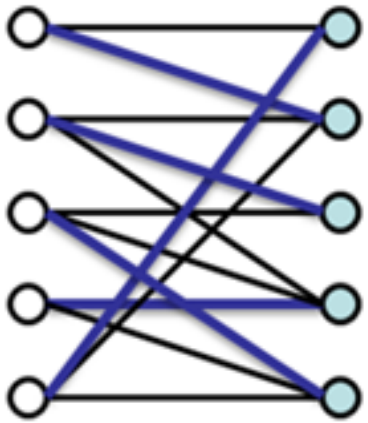


Figure 13.4.8 A maximum matching and minimum vertex cover .

Maximum Matchings and Minimum Vertex Covers

- **Proposition 13.4.7 [Weak Duality for Matchings]** Let M be a matching in a graph G, and let C be aa vertex cover of G. Then $|M| \leq |C|$.
- **Corollary 13.4.8 [Certificate of Optimality for Matchings]** Let M be a matching in a graph G, and let C be a vertex cover of G such that $|M| = |C|$ .Then M is a maximum matching and C is a minimum vertex cover.
- **Remark:** The converse of Corollary 13.4.8 does nit hold in general (see Exercises); however, it dies hold bipartite graphs.
- **NOTATION:** The neighbor set of a given graph subset W of vertices in a graph G is denoted $N_G(W)$ (instead of usual $N(W)$) when there is more than one graph involved.
- **Theorem 13.4.9 [König, 1931]** Let G be a bipartite graph G. Then the number od edges in a maximum matching in G is equal to the number if vertices in a minimum vertex cover of G.
  - ✓ Let $\{X,Y\}$be the vertex bipartition of bipartite graph G, and let C^*be a minimum vertex cover of G. Then C^* is the disjoint union of its set of X-vertices, $C^*_X = C^* \wedge X$, and its set of Y-vertices, $C^*_Y = C^* \wedge Y$, as illustrated in Figure 13.4.9.

Maximum Matchings and Minimum Vertex Covers

- ✓ Consider the bipartite subgraph G_1 of G graph G induced on the vertex bipartition $\{C^*_X, Y - C^*_Y\}$. Let W be any subset of C^*_X. If $|W| > |N_{G1}(W)|$, then there exist $w \in W$ such that
- ✓ $N_{G1}(W\text{-}w) = N_{G1}(W)$. But this would imply that $(C^*_X - \{w\}) \vee C^*_Y = C^*$ is a vertex cover of graph G, contradicting the minimality of C^*. Thus, the bipartite graph G_1 satisfies Hall's Condition, and by Hall's Theorem(Theorem13.4.3), G_1 has C^*_X-saturating matching M^*_1,with $|M^*_1| = |C^*_X|$.
- ✓ Next, let G_2 be the bipartite subgraph induced on the vertex bipartition $\{X - C^*_X, C^*_Y\}$. Then a similar argument applied to graph G_2 shows that it C^*_Y-saturating matching M^*_2, with $|M^*_2| = |C^*_Y|$. The edge set $M = M^*_1 \wedge M^*_2$ is clearly a matching in graph G, and $|M| = |C^*_X| + |C^*_Y| = |C^*|$. Thus, by Corollary 12.4.6, M is a maximum matching.

Maximum Matchings and Minimum Vertex Covers

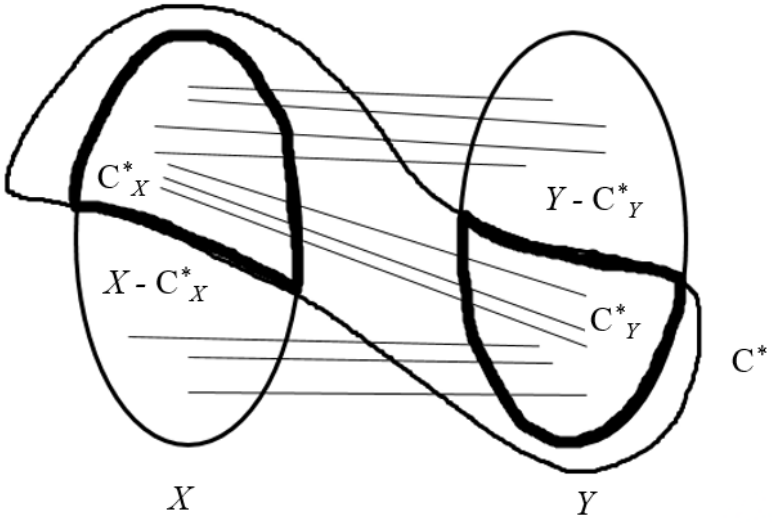


Figure 13.4.9 Minimum vertex cover $C^* = C^*_X \cup C^*_Y$

0-1 Matrices and the König-Egerváry Theorem

- An interesting interpretation of this last theorem involves *0-1 matrices*, which are matrices each of whose entries is 0 or 1.
- **Theorem 13.4.10 [König-Egerváry, 1931]** Let A be a 0-1 matrix. Then the maximum number of 1's in matrix A, no two of which lie in the same row or column, is equal to the minimum number of rows and columns that together contain all the 1's in A.
  - ✓ Let G be a bipartite graph with vertex bipartition$\{X,Y\}$, such that A is an adjacency matrix of graph G, where X is the set of vertices corresponding to the rows of matrix A, and Y is the set of vertices corresponding to the columns of matrix A. The result is applying Theorem 13.4.9(see Exercises).
- **Application 13.4.4** *The Bottleneck Problem:* Suppose that a manufacturing process consists of five operations that are performed simultaneously on five machines. The time in minutes that each operation takes when executed on each machine is given in the table below. Determine whether it is possible to assign the operations so that the process is completed within 4 minutes.

0-1 Matrices and the König-Egerváry Theorem

	M1	M2	M3	M4	M5
Op1	4	5	3	6	4
Op2	5	6	2	3	5
Op3	3	4	5	2	4
Op4	4	8	3	2	7
Op5	2	6	6	4	5

Let M be the 5x5 matrix whose ijth entry a_{ij} equals 1 if operation I takes no more than 4 minutes when performed on machine j, and equals 0 otherwise. Thus,

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

0-1 Matrices and the König-Egerváry Theorem

It is easy to check out that this matrix contains five 1's no two of which are in the same row or column, which implies that it is possible to complete the process within 4 minutes.

Summary of Equivalences Among Theorems

- The occurrence of various max-min pairs of problems and the similarities among many of the proofs in this chapter suggest strong connections among the theorems involved, some of which have already been established. In fact, any one of the following theorems can be used to prove the others.
  - Menger's Theorem (Theorem 13.3.18)
  - Max-Flow Min-Cut Theorem (Theorem 13.2.4)
  - König's Theorem (Theorem 13.2.4)
  - Hall's Theorem (Theorem13.4.3)
  - König-Egerváry Theorem (Theorem 13.4.10)
- Below is an outline of some of these connections.
  - Max-Flow Min-Cut → Menger (Theorem 13.3.5, 13.3.9, 13.3.17, 13.3.18)
  - Max-Flow Min-Cut → Hall (Theorem 13.4.3)
  - Hall → König (Theorem13.4.9)
  - König → König-Egerváry (Theorem13.4.10)
  - König-Egerváry → Hall (Theorem13.4.32)

Summary of Equivalences Among Theorems

- Menger → Hall (Exercise13.4.31)
- Menger → Max-Flow Min-Cut (Exercise13.3.25)