Hamilton Cycles, SDR's, Bipartite Graphs, Flows: Math 454 Lecture 14 (7/20/2017)

Cole Franks

July 29, 2017

Keller and Trotter Graph Theory chapter:

http://www.rellek.net/book/ch_graphs.html.

Keller and Trotter Network Flows chapter:

http://www.rellek.net/book/ch_networkflow.html.

You can read more about Eulerian trails/circuits in Brualdi 11.2 and more about Hamiltonian cycles/paths in Brualdi 11.3.

Contents

0.1 Eulerian circuit correction:

Theorem 0.1 (Eulerian circuits for looped *directed* multigraphs)**.** *A looped directed multigraph G with no isolated vertices has an Eulerian circuit if and only if*

$$d_+(v) = d_-(v)$$

for each vertex $v \in G$, and (different from 7/20 class!) the undirected version of G is connected.

1 Hamiltonian Cycles

1.1 A reason you might not have a Hamiltonian cycle

Theorem 1.1. *Suppose $G = (V, E)$ is a graph, and that there exists a subset $S \subset V$ of k vertices of G such that*

$$G(V \setminus S, E \setminus \{e : |e \cap S| > 0\})$$

has at least $k + 1$ connected components. Then G does not have a Hamiltonian cycle.

Another way to say this is that if you can remove some k vertices of G and all edges adjacent to them to create at least $k + 1$ connected components, then G cannot have a Hamiltonian cycle.

Proof. Suppose the set S of k vertices is such a subset. Let $C_1, \ldots C_l$, $l > k$, be the connected components in

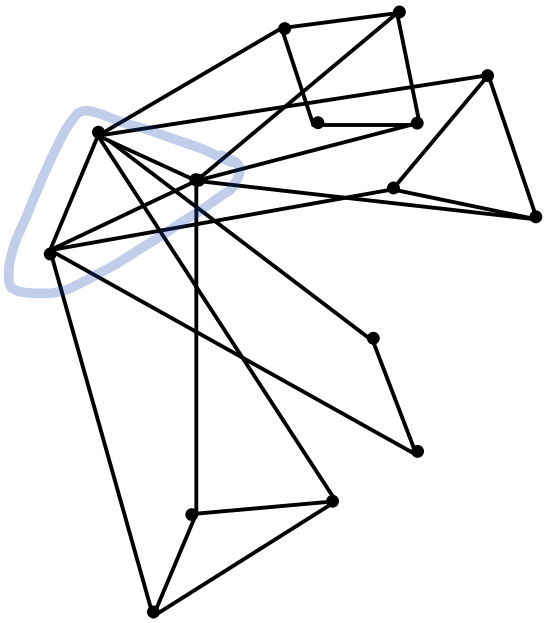
$$G(V \setminus S, E \setminus \{e : |e \cap S| > 0\}),$$

or G with S and all adjacent edges thrown out.

Suppose G had a Hamiltonian cycle C. Intuitively, C must visit each of the l connected components once (because each contains at least one vertex) and then return back to the component where it started. In between each of those visits, it has to visit at least one vertex of S. Thus, it visits at least l vertices of S.

Formally, let F be the edges of C contained entirely within S. The graph $G(S, F)$ must be a union of nonempty vertex-disjoint paths $P_1, \ldots P_t$ with the endpoints of each path adjacent in G to at most two components, C_i and C_j (possibly the same). Form a looped multigraph on l vertices with an edge between i and j for each path with one endpoint adjacent to i and one adjacent to j. This looped multigraph must contain a cycle on all the vertices, so it must have at least l edges, which means $t \geq l$. More than l nonempty vertex disjoint paths have at least l vertices, so $|S| \geq l$, a contradiction because $|S| = k < l$. □

Example 1. The following graph has no Hamiltonian cycle because deleting the circled three vertices and all adjacent edges leaves the remaining graph with four connected components.



1.2 Ore property

This is modified from class: in class, I forgot to mention that the Ore property only needs non-adjacent vertices to have high degree-sum.

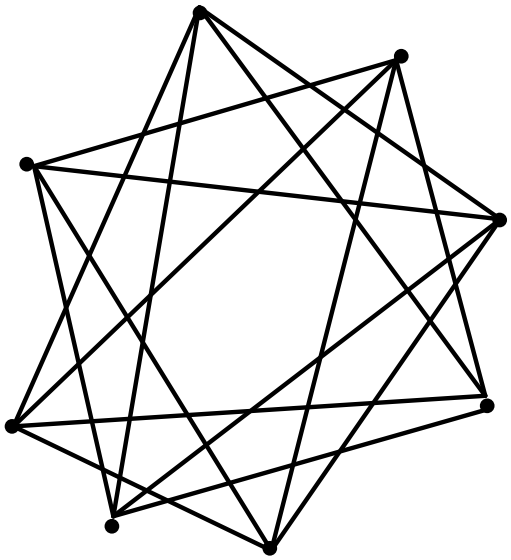
Definition 1.1. A graph G with n has the *Ore property* if

$$d(x) + d(y) \geq n$$

for every pair of distinct, *nonadjacent* vertices x and y of G.

Theorem 1.2 (Ore's Theorem)**.** *If G has the Ore property and has at least 3 vertices, then G has a Hamiltonian cycle.*

Example 2. The following graph must have a Hamiltonian cycle, even though I didn't bother finding it.



Proof. Suppose $G = (V, E)$ has the Ore property. We already saw in Homework 8 that G must be connected. So now we proceed with an algorithm of sorts.
The input of the algorithm is a path P.

Start: Replace P by a maximal path $y_1 \dots y_m$ containing P (one can do this by appending vertices adjacent to the endpoints if possible). Next, break into two cases.

Case 1. **($y_1y_m \in E$):** we can complete P to a cycle C with vertices $y_1, \dots, y_m$.

Case 1.a. **($m = n$:)** Done; output C.

Case 1.b. **($m < n$:)** Some vertex y_i of C is adjacent to a vertex x of G not on C. Replace P by a longer path, starting at x. That is, the path $xy_iy_{i+1} \dots y_my_1y_2 \dots y_{i-1}$. Go back to start.

Case 2, **$y_1y_m \notin E$:** P is a maximal path, so y_1 and y_m are only adjacent to vertices among $y_2, \dots y_{m-1}$. Say y_i, $i \in \{2, \dots, m-1\}$ is a *precedent* of y_1 if $y_1y_{i+1} \in E$.

Lemma 1.1. *There is a precedent of y_1 that is a neighbor of y_m.*

Proof. There are $d(y_1) - 1$ precedents of y_1 among $y_2, \dots, y_{m-2}$, and $d(y_m) - 1$ neighbors of y_m among $y_2, \dots, y_{m-2}$. If the precedents and neighbors were disjoint, then their union would have size $d(y_1) - 1 + d(y_m) - 1 \geq n - 2$ by the Ore property, but it would be contained in the $m - 3$-vertex set $y_2, \dots y_{m-2}$. However, $m - 3 < n - 2$, a contradiction. □

Since there is a precedent y_i of y_1 that is a neighbor of y_m, replace P by the path $C = y_1y_2 \dots y_iy_my_{m-1}y_{m-2}, \dots y_{i+1}$ (which has adjacent endpoints) and go back to Case 1.

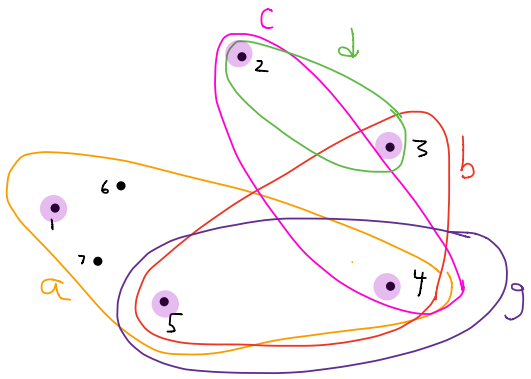
This algorithm terminates because the only way to get back to the start is through case 1.b., and each time we do so we add a vertex to P. Eventually we must enter case 1.a. and output a Hamiltonian cycle. □

2 Systems of Distinct representatives

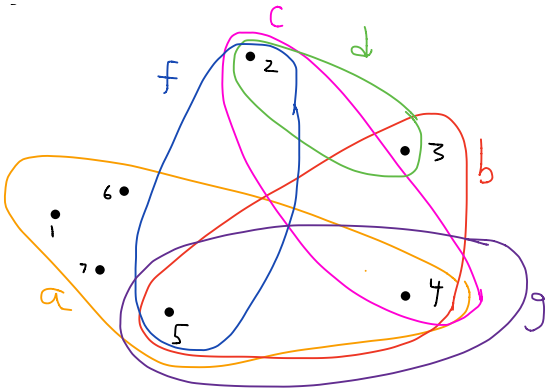
Suppose there are a collection of n candidates and m jobs. Each job j has a set of candidates B_j that are qualified for the job j. Our objective is to hire one qualified candidate for each job. We want to find a *system of distinct representatives (SDR for short)*, or a list of n distinct candidates

$$(c_1, \ldots, c_m) \text{ such that } c_j \in B_j \text{ for all } j \in [m].$$

Example 3. 7 candidates [7] and 5 jobs $\{a, b, c, d, g\}$. Candidate 1 for job a, candidate 5 for job b, candidate 4 for job g, candidate 3 for job d, and candidate 2 for job c is an SDR.



Example 4. If the collection of jobs looked instead like the following, we would be out of luck, because 5 jobs b, c, d, f, g only have 4 qualified candidates between them.



Let's describe this issue fully. If we have a family $B_1, \dots B_m$ of sets, we will never be able to find a system of distinct representatives if there is a subset $\{i_1 < i_2 < \cdots < i_k\} \subset [m]$ with

$$|B_{i_1} \cup B_{i_2} \cup \cdots \cup B_{i_k}| < k.$$

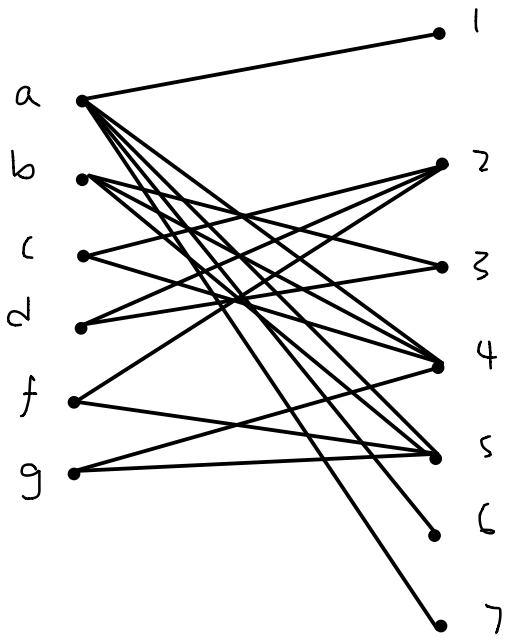
2.1 Bipartite graphs

We can restate this in terms of bipartite graphs.

Definition 2.1. A graph $G = (X \cup Y, E)$ is *bipartite* if X and Y are disjoint sets such that every edge e intersects both X and Y.

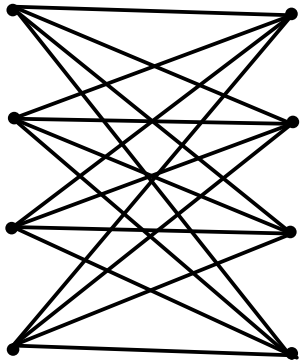
To get a bipartite graph from the jobs example, let $X = \{jobs\}$ and $Y = \{candidates\}$, and let jc be an edge if candidate c is qualified for job j.

Example 5. The following is a bipartite graph of the set family of jobs from Example 4.



A *complete bipartite graph* on X and Y is a bipartite graph $G = (X \cup Y, E)$ where $E = \{\{x, y\} : x \in X, y \in Y\}$. That is, all possible edges between X and Y are present. If $X = [m]$ and $Y = [n]$, the complete graph on X and Y is denoted $K_{n,m}$.

Example 6. A $K_{4,4}$:



Definition 2.2. In general, the bipartite graph of a family of subsets $B_1, \ldots, B_m$ of Y is a bipartite graph

$$G = ([m] \cup Y, E)$$

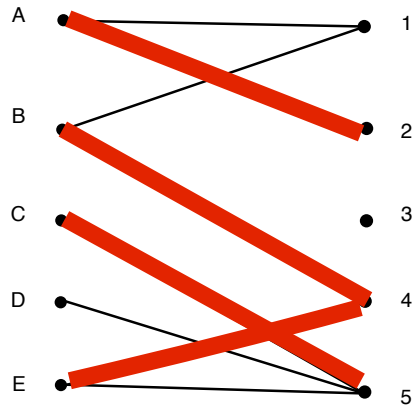
where $\{i, y\} \in E$ if and only if $y \in M_i$.

Remark 2.1. Annoyingly, Y could intersect $[m]$. We won't worry about this. Whenever elements of Y seem to appear in $[m]$, just assume those elements of Y are clones that aren't "really" in $[m]$.

2.2 Matchings

Definition 2.3. A *matching* in a bipartite graph $G = (X, Y)$ is a collection of *disjoint* edges in G. An *complete matching* from X to Y is a matching M such that X is contained in the union of the edges of M. That is, every vertex of X gets "matched."

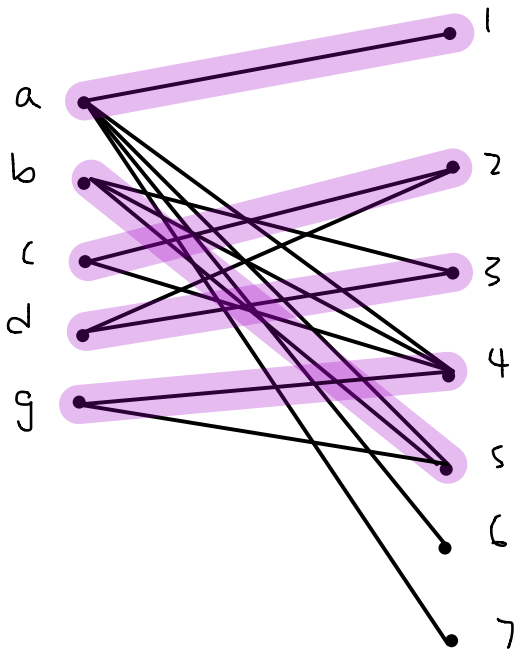
Remark 2.2. The edges of a matching are *disjoint*, so no two edges in the matching are incident to the same vertex! A NON-MATCHING:



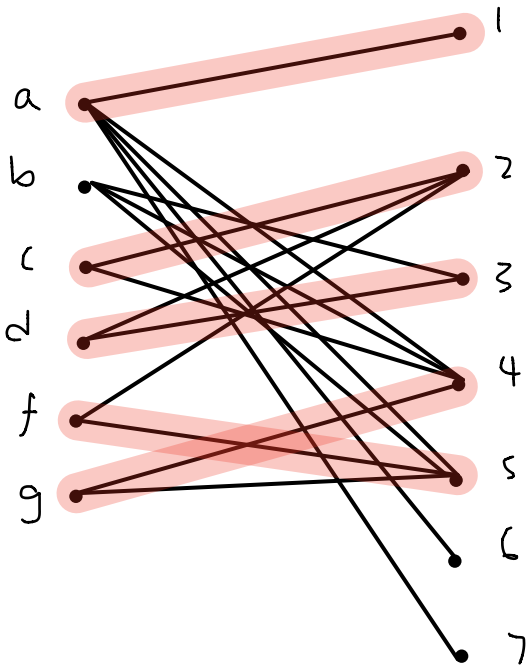
The red edges are *NOT A MATCHING* because two of the edges are incident to 4. The edges $C5$ and $E4$ are not the reason this is not a matching. It's fine if they cross when drawn; this is just a symptom of how we chose to draw the graph.

A system of distinct representatives in a family of subsets $B_1, \ldots, B_m \subset Y$ is a collection of disjoint edges $\{1, y_1\}, \ldots \{m, y_m\}$ in the bipartite graph of the set family.

Example 7. The system of distinct representatives from Example 3 corresponds to the matching in the following bipartite graph:



Example 8. The following is the best we can do in the set family of jobs from Example 4. It is an incomplete matching. Of course, there are other incomplete matchings of size 5.



To summarize, SDR's and complete matchings are one and the same:

Fact 2.2.1. A family of subsets $B_1, \ldots, B_m \subset Y$ has a system of distinct representatives if and only if the bipartite graph of the family has a complete matching of $[m]$ to Y.

3 Flows

Definition 3.1. A *network* $N = (G, c, s, t)$ consists of a

- Directed graph $G = (V, E)$,
- A distinguished vertex $s \in V$ called the *source* and a different distinguished vertex $t \in V$ called the *sink*.
- A function $c : E \to \mathbb{R}^+$ assigning nonnegative numbers to the edges. If $(x, y) \in E$, $c(x, y)$ is called the capacity of E.

Definition 3.2. A flow on a network $N = (G, c, s, t)$ is another function $f : E \to \mathbb{R}^+$ assigning nonnegative numbers to the edges of G satisfying

- $f(x, y) \leq c(x, y)$ for all $(x, y) \in E$.
- $f(x, y) = 0$ for $(x, y) \notin E$.
- For vertices $v \in V \setminus \{s, t\}$, that is, vertices other than the sink and source, the total flow into v is equal to the total flow leaving v. Precisely,

$$\sum_{(x,v)\in E} f(x, v) = \sum_{(v,x)\in E} f(v, x)$$

The value of the flow, denoted $value(f)$, is defined to be

$$value(f) = \sum_{(s,y)\in E} f(s, y) - \sum_{(x,s)\in E} f(x, s).$$

In words, it is the flow leaving the source minus the flow leaving the source.

Definition 3.3. A *cut* in a network is a set S containing the source but not the sink. The *capacity* of a cut S is denoted $\text{cap}(S)$, and is defined to be

$$\text{cap}(S) = \sum_{(x,y)\in E(S,V\setminus S)} c(x, y),$$

where $E(S, V \setminus S)$ denotes all edges (x, y) such that $x \in S$ and $y \in V \setminus S$.

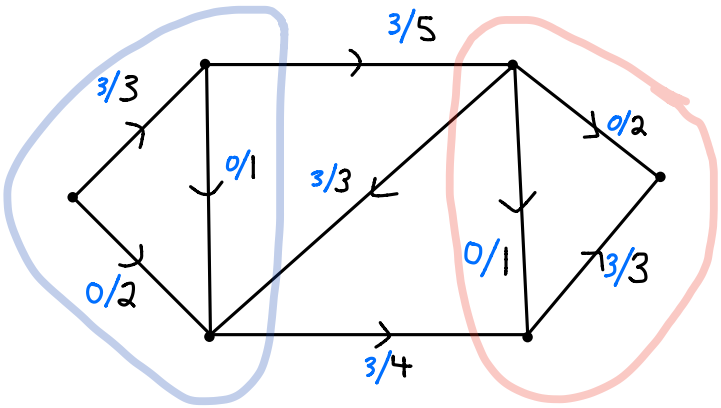
Fact 3.0.1. Let f be a flow. If S is a cut, then

$$value(f) = \sum_{(x,y)\in E(S,V\setminus S)} f(x, y) - \sum_{(x,y)\in E(V\setminus S,S)} f(x, y).$$

In other words, the value of the flow is the total flow leaving S minus the total flow entering S. In particular,

$$value(f) \leq \text{cap}(S).$$

Example 9. The following is a network with capacities in black. There is a flow of value 3 on the network (the blue numbers), and a cut circled in light blue of capacity 6. The source is the left most vertex and the sink is the rightmost.



3.1 Ford Fulkerson and Max-flow-Min-cut

Theorem 3.1. *The maximum of the value of a flow in G is equal to the minimum capacity of a cut in G.*

We only prove this for integer capacities, which covers all the cases we need. Thus, in the following proof, assume $c(x, y)$ is an integer.

Proof. We build a flow iteratively. Initially, let f be the zero flow given by $f(x, y) = 0$ for all $(x, y) \in E$.

Define the directed graph R, called the *residual graph*, to contain the edge (x, y) if $f(x, y) < c(x, y)$ or $f(y, x) > 0$. That is, there is still "room" to increase the flow on that edge, or we could decrease the flow in the opposite direction.

If there exists a path P given by$x = x_0, x_1, \ldots x_m = t$ in R from s to t, alter the all the corresponding flows P to increase the value of the flow in the following manner.

- If (x_i, x_{i+1}) is an edge of G (it goes the "right way" in G) then $f(x_i, x_{i+1}) \geq c(x_i, x_{i+1}) - 1$, so we can replace $f(x_i, x_{i+1})$ by $f(x_i, x_{i+1}) + 1$.
- If (x_{i+1}, x_i) is an edge of G (it goes the "wrong way" in G) then $f(x_{i+1}, x) \geq 1$, so we can replace $f(x_{i+1}, x_i)$ by $f(x_{i+1}, x_i) - 1$.

Update R, and repeat this process until you cannot reach t from s in R.

You can check that after each iteration, this is still a flow, has a value at least 1 higher than before, and all flows are integers.

This process must terminate in at most the maximum value of the flow, which is at most the sums of the capacities of all the outgoing edges from s, which is finite.

Once this process gets stuck, let S be everything reachable in R from s. It cannot contain t, or else we wouldn't have gotten stuck. Then S is a cut, and $\text{cap}(S) = value(f)$. This is because any edge from S to $V \setminus S$ in G must have flow equal to its capacity, or it would also be in R and thus one more vertex would be reachable from S. Similarly, any edge from $V \setminus S$ to S in G must have zero flow. This means we have achieved a flow with the same value as the capacity of a cut. By Fact 3.0.1 we already know that the value of every flow is less than the capacity of every cut, so this flow must attain the maximum value. □

Remark 3.1. A couple of things to note:

Integrality This proof tells us that if the capacities are integers, then there is a maximum flow with integer flows on every edge.

Infinite capacities: The proof would also work even if some capacities were infinite, that is, there's no limit on the capacity of some edges. All you need for the algorithm to terminate is that the total value of the flow is finite, or equivalently, there's a cut with finite capacity.

This algorithm still works if the capacities are rational (just multiply the capacities by a large enough integer), but if the capacities are irrational it doesn't necessarily converge.

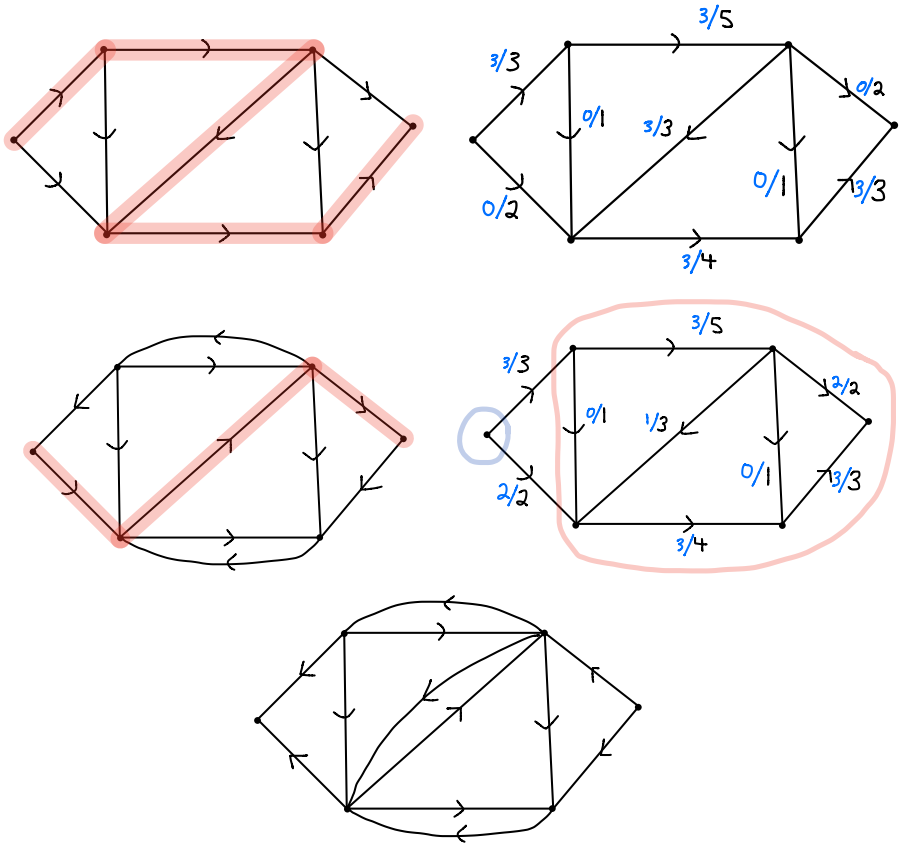


Figure 1: .
The top left image is the residual graph for the zero flow, along with a path from s to t. top right is the flow (of value 3) when updated along that path. The middle left is the residual graph after the flow has been updated along with a path, and the middle right is the flow when updated along that path. The bottom image is the residual graph for the flow (of value 5) in the fourth, and you can see that there is no path from source to sink. The only vertex reachable from the source is the source itself; this gives a cut consisting of the source of capacity 5, which is shown in the middle right image.