CS 787: Advanced Algorithms

Flows and Cuts

Instructor: Dieter van Melkebeek

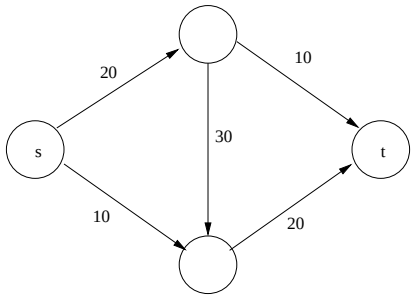
This lecture covers the construction of optimal flows and cuts in networks, their relationship, and some applications. It paves the way to our subsequent treatment of linear programming. The network flow and cut algorithms do not follow any of the paradigms we discussed so far, not do they represent a paradigm of their own. The applications we will present illustrate the paradigm of efficient reductions.

1 Concepts

We define the notions of *network*, *flow*, and *cut*.

Definition 1 (Network)**.** *A network consists of a directed graph, $G = (V, E)$, a capacity function, $c : E \mapsto [0, \infty)$, and two distinguished vertices: a source s and a sink t. The source vertex has only outgoing edges and the sink has only incoming edges.*

Example: The following figure shows a sample network. The capacities are shown next to each edge and the source and sink vertices are labeled as s and t.



⊠

The edges in a network can be thought of as pipes that can carry "traffic" of some kind: water, oil, electricity, cars, bits, etc. The capacity of an edge indicates the maximum amount of traffic that the edge can carry per unit of time. The vertices represent connections between pipes. The source is the only place where traffic can enter the system, and the sink where it leaves the system. Everywhere else traffic is conserved. The setting is captured by the following definition of "flow," which is the technical term for "traffic."

Definition 2 (Flow)**.** *Given a network $G = (V, E)$, a flow is a function $f : E \mapsto [0, \infty)$ such that:*

1. $(\forall e \in E)\, f(e) \leq c(e)$
2. $(\forall v \in V \setminus \{s, t\})\ \sum_{(u,v)\in E} f(u, v) = \sum_{(v,u)\in E} f(v, u)$

A flow function f maps each edge to the amount of traffic it carries. Requirement 1 in the preceding definition represents the capacity constraints of teh network: no edge can carry more flow that its capacity. Requirement 2 expresses the conservation constraints: all vertices except the source and sink must have equal inflow and outflow. The source is allowed to have net outflow, and the sink may have net inflow.

A flow specifies some way of traffic being routed from the source vertex to the sink vertex observing the capacity of the network. The value, ν, of a flow quantifies the amount of traffic being routed from source to sink. One way to formalize this is as the total amount flowing on the outgoing edges of the source:

$$\nu(f) = \sum_{(s,u)\in E} f(s,u).$$

The definition of ν seems rather arbitrary. We could as well have defined it as the total amount of flow into the sink. Both definitions are actually equivalent. In fact, for every cut in the network that separates the source s from the sink t, the net amount of flow that crosses the cut from the source to the sink equals ν.

We formalize the notion of cut as follows.

Definition 3 (st-Cut)**.** *An st-cut in a network is any partition (S,T) of the vertices into two sets such that $s \in S$ and $t \in T$.*

We leave the following as an exercise:

Proposition 1. *For every st-cut (S,T),*

$$\nu(f) = \sum_{e\in E\cap S\times T} f(e) - \sum_{e\in E\cap T\times S} f(e). \tag{1}$$

Note that the first term on the right-hand side of (1) represents the total flow through the edges that cross the cut from the source-side S to the sink-side T, and the second term the total flow through the edges that cross the cut in the opposite direction. For $S = \{s\}$, the second term in (1) vanishes, and we obtain our definition of ν; for $T = \{t\}$ we obtain our alternate definition.

The idea of capacity can be extended from single edges to cuts. In particular, the capacity of a cut is defined as:

$$c(S,T) = \sum_{e\in E\cap S\times T} c(e)$$

Note that only the edges that cross the cut from the source-side to the sink-side contribute to the capacity of the cut; the edges that cross the cut in the opposite direction do not count.

2 Duality

A simple observation regarding network flow leads to the idea of weak duality. Here we make this observation and also state strong duality, which will be proven later. Both are closely related to duality in linear programming, and will be revisited when we cover the latter.

With regard to the structures presented previously, the following observation may be made:

Proposition 2. *Given any flow f and any st-cut (S,T) in a network G, the value of the flow cannot exceed the capacity of the cut:*

$$\nu(f) \leq c(S,T).$$

This observation follows from Proposition 1. This is because the first term on the right-hand side of (1) is upper bounded by the capacity of the st-cut (S,T), and the second term is nonnegative. Moreover, the equality $\nu(f) = c(S,T)$ holds iff the following conditions hold simulteneously:

1. All "forward" edges from S to T are used at full capacity.
2. No "backward" edges from T to S carry flow.

Since Proposition 2 holds true for any flow and any st-cut, the maximum value of a flow is upper bounded by the minimum capacity of an st-cut. This relationship is known as *weak duality* for network flow problems:

$$MAXFLOW \leq MINCUT$$

In fact, we will later show that equality always holds:

$$MAXFLOW = MINCUT$$

This is known as the (strong) duality of network flow, and we will prove it in a constructive way.

3 Path Augmentation

There are many algorithms for finding a flow of maximum value in a given network. One class of algorithms start with a zero flow, and iteratively augment the current flow by selecting a path P from the source to the sink and pushing as much additional flow along that path as possible without violating any of the capacity constraints. This process is known as path augmentation.

If we only consider paths in the original network, the process may get stuck at a suboptimal flow. See the presentation from class for an example. We need to allow path augmentations that undo (part of) the flow through some edges of the given network. In order to facilitate this process, we define the *residual network*.

Definition 4 (Residual network)**.** *Given a flow f on a network $G = (V,E)$, the residual network G_f is a network on the same vertices and with the same source and sink, but with a possibly different edge set. That edge set can contain both edges from the original network G as well as reverse edges. Specifically,*

$$G_f = (V, E_f)$$

where

$$E_f = \{e \in E \mid f(e) < c(e)\} \cup \{e \in E^{-1} \mid f(e) > 0\}$$

The capacities of the residual network are calculated by subtracting the flow on an edge from the capacity of that edge in G (if that edge is in E), or by the flow of e^{-1} if e is the reverse of an edge in E.

$$c_f(e) = \begin{cases} c(e) - f(e) & e \in E_f \cap E \\ f(e^{-1}) & e \in E_f \cap E^{-1} \end{cases}$$

The edges in a residual network either indicate flow that is still under an original edge's capacity, or flow that is in use and can be reverted. We then define an *augmenting path* as follows.

Definition 5 (Augmenting path)**.** *Given a flow f in a network G with source s and sink t, an augmenting path P is a path in G_f from s to t.*

Note that the residual network only contains edges with positive residual capacity. Therefore, the existence of an augmenting path indicates that the value of the flow can be increased without violating capacity limitations by increasing flow pushed along edges of G with capacity remaining and/or decreasing flow along a currently used edge.

The idea then is to keep finding augmenting paths, using each to its capacity, recalculating the residual network, and repeating until no more augmenting paths exist. This approach is known as the Ford-Fulkerson scheme. The term "scheme" refers to the unspecified nature of the criterion for selecting the augmenting path in case multiple exist.

FORD-FULKERSON SCHEME($G = (V, E), s, t \in V$)
$f \leftarrow 0$
While there exists an augmenting path in G_f:
  select such a path P
  $f_P \leftarrow$ maximum additional flow that can be pushed through P
  $f \leftarrow f + f_P$
Return f

Naturally, two questions arise about this scheme: "Does it halt?" and "If it halts, is it guaranteed to return a maximum flow?". We will start by showing the latter. This will provide proof of partial correctness of the Ford-Fulkerson scheme.

Partial correctness.

Theorem 1. *The following are equivalent for any flow f in a network G:*
(1) $\nu(f)$ is maximum.
(2) f has no augmenting path.
(3) There exists a cut (S, T) in G of capacity $\nu(f)$.

Proof. We argue using cyclic implications.

(1) $\Rightarrow$ (2). As shown, if there is an augmenting path P in G_f, then there is as yet unused capacity in all the edges of P, and we can push a positive amount of flow along these edges, increasing the value of our flow by that amount. Hence, the presence of an augmenting path implies that f is not maximal. The contrapositive proves our claim.

(2) $\Rightarrow$ (3). Consider the set

$$S = \{u \in V \mid u \text{ is reachable from } s \text{ in } G_f\},$$

and let $T = V \setminus S$. Note that $s \in S$ because of the trivial path from s to s in G_f. The hypothesis that f has no augmenting path means that $t \in T$. Thus, (S, T) forms an st-cut.

Moreover, every edge $e = (u, v) \in E \cap S \times T$ has to be used at full capacity under f. Otherwise, e would be an edge in G_f, and since u is reachable from s in G_f by the definition of being in S, so

would v, which contradicts v being in T. Similarly, every edge $e = (v, u) \in E \cap T \times S$ cannot be used by f. Otherwise, the reverse edge $e^{-1} = (u, v) \in S \times T$ would be present in G_f, which leads to the same contradiction as before. Thus,

$$f(e) = \begin{cases} c(e) & \text{if } e \in E \cap S \times T \\ 0 & \text{if } e \in E \cap T \times S \end{cases}$$

As mentioned after Proposition 2, this implies that $\nu(f) = c(S, T)$.

(3) $\Rightarrow$ (1). This can be proven by simple application of weak duality, $MAXFLOW \leq MINCUT$. Since our flow value, $\nu(f)$ is equal to the capacity of some cut by (3), we know that its value is maximum. □

Note the constructive nature of the proof of Theorem 1: Given a maximum flow f, it shows how to construct a minimum cut in linear time.

Two corollaries are implied by Theorem 1.

Corollary 1 (Strong duality)**.** *The relationship*

$$MAXFLOW = MINCUT$$

holds for all networks.

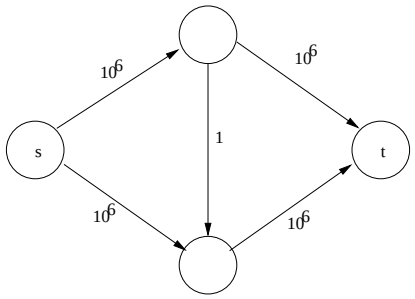
This follows from the implication (1) $\Rightarrow$ (3). In fact, the proof of Theorem 1 yields a linear-time algorithm to construct minimum cut given a maximum flow.

Corollary 2 (Partial correctness of Ford-Fulkerson)**.** *If the Ford-Fulkerson halts, it returns a maximum flow.*

This follows from the invariant that f always represents a valid flow, and from the implication (2) $\Rightarrow$ (1).

Termination. So we now know that if the scheme halts, we will get a maximum flow. We now turn to the question of whether the scheme will actually halt, as desired. Observe that if all the capacities in G are integer-valued, then at every point in time the residual capacities, the flow f_P, and he flow f are integer-valued, and the Ford-Fulkerson scheme will increase the flow by a minimum of 1 at each iteration. Hence, the number of iterations is bounded by the value of the maximum flow. Therefore, if all capacities are integer, the scheme is guaranteed to halt. By scaling, the same applies when all capacity ratios are rational.

However, there are examples where the capacity ratios are irrational for which there exist infinite sequences of path augmentations that does not even converge to a maximum flow. The examples may be contrived, but hint that the running time on less-contrived examples with integer capacities may be bad. Indeed, the value of the maximum flow is not a desirable bound, as in the case of a network like the one shown below. If the scheme repeatedly makes a poor choice for the augmenting path, it improves the flow by only one at each iteration. If the maximal flow were a value like two million as shown, what could simply be a two iteration job may be 6 orders of magnitude worse!



This implies that we need to consider more carefully how the augmenting paths are chosen. We present two good options, both of which generalize the optimal choice of path augmentations in the example.

- *Path of largest residual capacity.*
  One alternative is to choose an augmenting path for which we can increase our flow by the maximum amount. We leave as an exercise to find such a path in linear time. Assuming all capacities are integer, this method causes the Ford-Fulkerson scheme to run for $O(m \log(nC))$ iterations, where $C = \max_{e \in E} c(e)$. The resulting running time is $O(n + m^2 \log(nC))$. Note that the bound on the number of iterations is polynomial in the bit-length of the input, but depends on the values of the numbers involved.

- *Path with the smallest number of edges.*
  Another alternative is to choose an augmenting path with the least number of edges. This can also be done in linear time, using a breadth-first search. The number of iterations for the scheme using this method is $O(nm)$ (see the handout), resulting in an overall running time of $O(nm^2)$. Note that this alternative is always guaranteed to halt, no matter what the capacities are. In fact, the bound on the number of iterations is polynomial in the size of the network and does not depend on the bit-length of the numbers involved. A polynomial-time algorithm of that type is called *strongly polynomial-time*.

There are other approaches to solving network flow problems, and some are more efficient than the augmenting path solution presented here. The best algorithms to date are (a) one that works for integer capacities and runs in time $O(m \cdot \min(n^{2/3}, \sqrt{m}) \cdot \log(n^2/m) \cdot \log(C))$, and (b) one that works for all capacities and runs in time $O(nm)$. We still do not know anything close to linear time. A recent breakthrough was the design of algorithms that produce an ϵ-approximate maximum flow in time $O(n + m^{1+o(1)}/\epsilon^2)$.

4 Applications

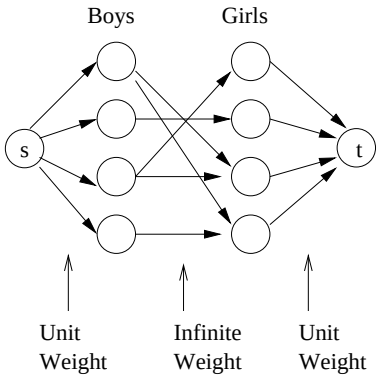
We now present some interesting problems that can be reduced to maximum flow or minimum cut.

4.1 Bipartite Matching

Given: biparte graph $G = (V, E)$ with left vertex set L and right vertex set R.

Goal: find a matching M of maximum size, i.e., a set $M \subseteq E$ such that no two distinct edges in M intersect.

A perfect matching is one where all vertices are matched. The problem can be thought of as matching n boys and n girls into pairs in which the boy and girl both like each other. It can be reduced to a network flow problem if the set of boys L and the set of girls R are graphed so that each edge connects a boy b to girl g with infinite capacity if they like each other (unit capacity would actually be fine for now, but we'll see soon why infinite capacities are more suitable). We create a source s with unit-capacity edges to all boys, and a sink t with unit-capacity edges from all girls. We call the resulting network N. An example of such a network is shown below.



Proposition 3. *There is a one-to-one and onto correspondence between integer flows f in N and matchings M in G. Moreover, $\nu(f) = |M|$.*

Proof. Given a matching M, for each edge $\{b, g\}$ in M, consider a flow of one unit along the path $s \to b \to g \to t$. Those are all valid flows of unit value. As the edges in M do not have any vertices in common, superimposing them yields a valid flow f of value $|M|$.

Conversely, given an integer flow f, the set of middle edges that carry a positive amount of flow form a matching M. Indeed, no two middle edges that carry flow can share a boy b, as that would mean b receives at least 2 units of flow from s. Similarly, no two middle edges that carry flow can share a girl g. It also follows that middle edges do not carry more than one unit of flow. As the middle edges form a cut, there have to be $\nu(f)$ that carry flow, so $|M| = \nu(f)$.

We leave it as a exercise that both mapping are each other's inverse. The proposition follows. □

If we solve the network flow problem using path augmentation, each augmentation will increase the value of the flow by one. Thus, there are at most n augmentations, each taking $O(m)$ work, for a total running time of $O(nm)$.

It is often interesting to investigate the meaning of the dual problem. In this case, an st-cut has a finite value iff no edge $(b, g) \in E$ is cut, i.e., $E \cap S \times T = \emptyset$, or equivalently, for each $(b, g) \in E$, either $b \in T$ or $g \in S$ or both. Yet another way of stating this is that $C \doteq (L \cap T) \cup (R \cap S)$ forms a vertex cover of G. We have the following propostion.

Proposition 4. *There is a one-to-one and onto correspondence between st-cuts (S, T) of finite capacity in N and vertex covers C in G. Moreover, $c(S, T) = |C|$.*

As we can find a minimum cut in polynomial time, the proposition implies that we can find a minimum vertex cover in a bipartite graph in polynomial time. This stands in contrast to finding minimum vertex covers in general graphs, which is NP-hard.

Finally, suppose there does not exist a perfect matching. This means there exists an st-cut (S, T) such that

$$c(S, T) = |L \cap T| + |R \cap S| < n.$$

Now, observe that

$$|L \cap T| = n - |L \cap S|$$

(since there are exactly n boys), and

$$\Gamma(L \cap S) \subseteq R \cap S,$$

where Γ means "every girl liked by a boy in this set" (since otherwise the cut would have infinite capacity). It follows that $|\Gamma(L \cap S)| < |L \cap S|$, that is, the set of girls liked by a boy in S is smaller than the set of boys in S. This is the reason why there is no perfect matching. We have argued *Hall's Theorem*: a bipartite graph with left vertex set L of size n and right vertex set R of size n has no perfect matching iff there exists a set $A \subseteq L$ such that $|\Gamma(A)| < |A|$.

4.2 Project Selection

Given:

- a set $L \doteq [n]$ of projects with benefit b_i for $i \in [n]$
- a set $R \doteq [m]$ tools with cost c_j for $j \in [m]$
- bipartite graph with left vertex set L and right vertex set R indicating whether a project i requires a tool j

Goal: Find a selection A of projects that maximizes the net revenue

$$\sum_{i \in A} b_i - \sum_{j \in \Gamma(A)} c_j.$$

The fact that we need to select a subset A of the projects smells like a cut problem. However, we need to maximize an objective. In order to reduce to a min-cut problem, first note that

$$\max_A \left\{ \sum_{i \in A} b_i - \sum_{j \in \Gamma(A)} c_j \right\} = - \min_A \left\{ \sum_{j \in \Gamma(A)} c_j - \sum_{i \in A} b_i \right\}$$

Moreover, observe that

$$- \sum_{i \in A} b_i = - \sum_{i=1}^{n} b_i + \sum_{i \notin A} b_i$$

and that the first term $\sum_{i=1}^{n} b_i$ is a constant.

Now, we create a network as follows. The source s has an edge to each project i with capacity b_i, and each tool j has an edge to the sink t with capacity c_j. Finally, we orient the edges of the bipartite graph from the projects to the tools, and assign those infinite capacities. See 1 for a sample graph.

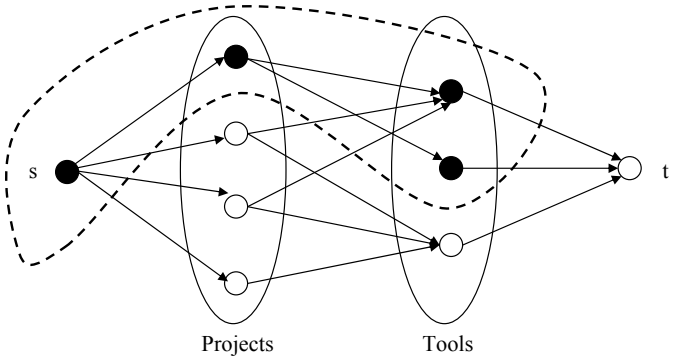


Figure 1: An example graph for the project selection problem

Similar to the setting of bipartite matching, there is a one-to-one and onto correspondence between st-cuts (S, T) of finite capacity and selections of projects A and tools B such that $\Gamma(A) \subseteq B$: $A = L \cap S$ and $B = R \cap S$. Moreover, the capacity of the cut equals

$$c(S,T) = \sum_{i \notin A} b_i + \sum_{j \in B} c_j. \tag{2}$$

Cuts of minimum capacity will have $B = \Gamma(A)$, in which case the right-hand side of (2) coincides with our objective function modulo the constant term $\sum_i b_i$. As this constant does not affect which selection A realizes the minimum, we can solve project selection by finding a minimum cut in the network we constructed. Thus, we can solve project selection in time $O(nm)$, where m denotes the number of edges of the bipartite graph.

Note that the minimum vertex cover problem in a bipartite graph can be cast as a project selection problem, where project $i \in L$ corresponds to not taking up i in the vertex cover C, and tool $j \in R$ corresponds to taking up j in the vertex cover C. In fact, this way we can reduce the minimum weighted vertex cover problem in bipartite graphs to min-cut, and solve it in polynomial time.

4.3 Image Segmentation

Given:

- A grid of pixels i.
- For each pixel i, a likelihood f_i that i belongs to the foreground, and a likelihood b_i that i belongs to the background
- For each pair $\{i, j\}$ of neighboring pixels, a penalty s_{ij} for separating i and j. For simplicity, we assume that $s_{ij} = s_{ji}$.

Goal: Find a partition of the pixels into a foreground F and a background B such that

$$\sum_{i \in F} f_i + \sum_{j \in B} b_j - \sum_{i \in F, j \in B, \{i,j\} \in E} s_{ij} \tag{3}$$

is maximized.

We consider the pixels as the vertices of a grid graph $G = (V, E)$. The value f_i can be taken as the intensity of pixel i, and b_i its complement.

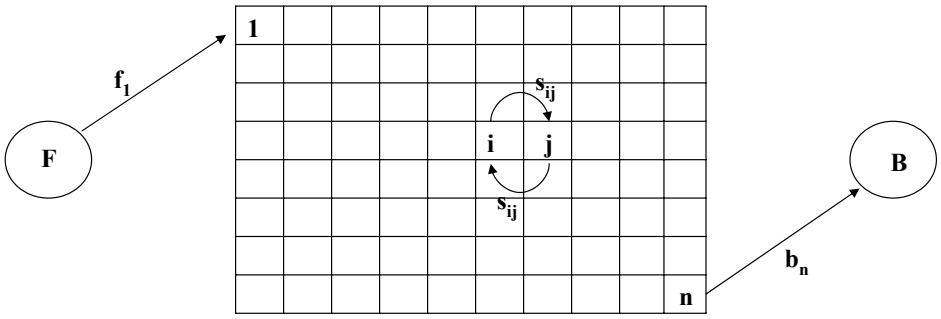


Figure 2: Image segmentation problem

As illustrated in Figure 2, we introduce two directed edges between each pair $\{i, j\} \in E$ so that both directions are covered, and we set the capacity of each of them to be s_{ij}. We create also a source node s and a sink node t. Similar to the previous application, we introduce edges from s to each pixel i with capacity f_i, and edges from each pixel j to t with capacity b_j. The capacity of an st-cut (S, T) equals

$$\sum_{i \in T} f_i + \sum_{j \in S} b_j + \sum_{\{i,j\} \in E, |\{i,j\} \cap S| = 1} s_{ij}.$$

By similar manipulations as in our discussion of project selection, it follows that a minimum st-cut (S, T) corresponds to an optimal partition (F, B) of the image maximizing our objective function (3), where the correspondence is given by $S = F \cup \{s\}$ and $T = B \cup \{t\}$.