The Ford–Fulkerson Algorithm

Math 482, Lecture 26

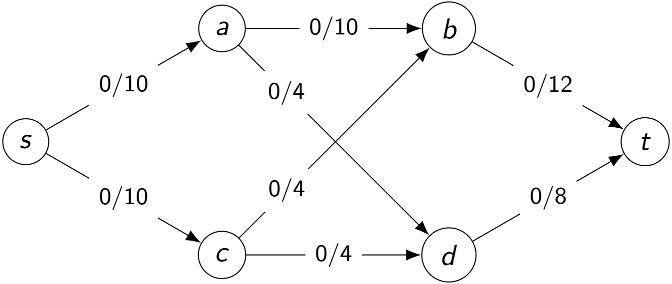
Misha Lavrov

April 6, 2020

A summary of the last lecture

In the previous lecture, we found a high-value flow in a network by starting with the zero flow and repeating the following procedure:

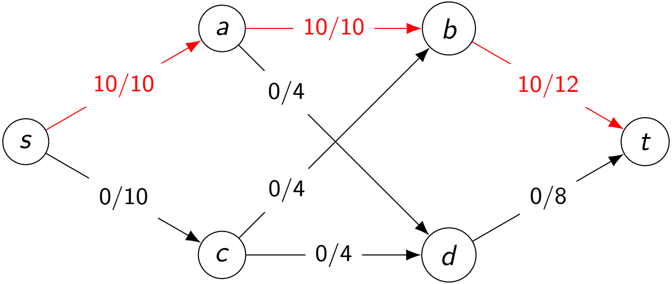
1. Find an augmenting path.
2. Use it to augment the flow as much as possible.



A summary of the last lecture

In the previous lecture, we found a high-value flow in a network by starting with the zero flow and repeating the following procedure:

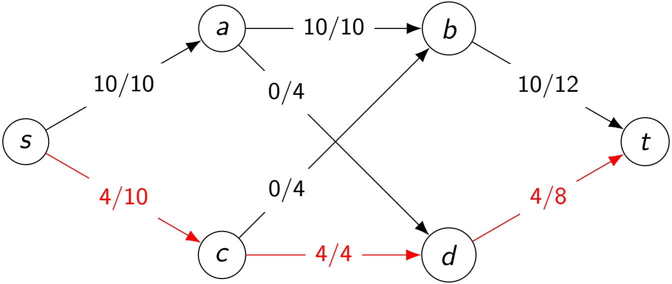
1. Find an augmenting path.
2. Use it to augment the flow as much as possible.



A summary of the last lecture

In the previous lecture, we found a high-value flow in a network by starting with the zero flow and repeating the following procedure:

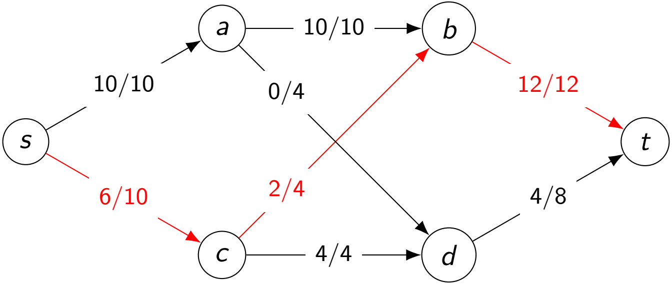
1. Find an augmenting path.
2. Use it to augment the flow as much as possible.



A summary of the last lecture

In the previous lecture, we found a high-value flow in a network by starting with the zero flow and repeating the following procedure:

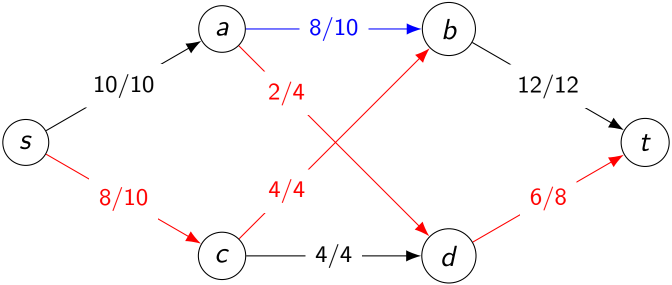
1. Find an augmenting path.
2. Use it to augment the flow as much as possible.



A summary of the last lecture

In the previous lecture, we found a high-value flow in a network by starting with the zero flow and repeating the following procedure:

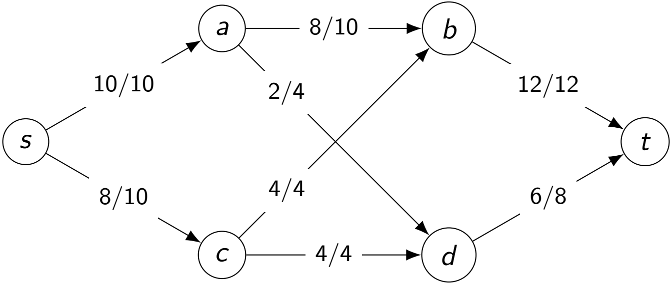
1. Find an augmenting path.
2. Use it to augment the flow as much as possible.



A summary of the last lecture

In the previous lecture, we found a high-value flow in a network by starting with the zero flow and repeating the following procedure:

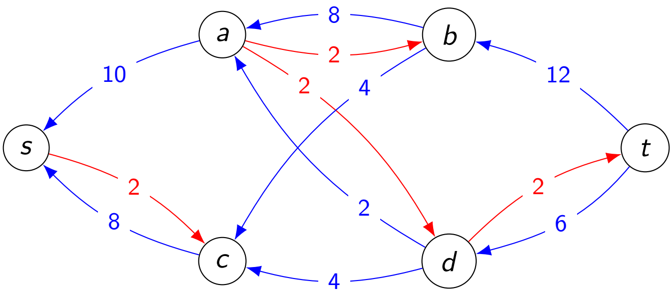
1. Find an augmenting path.
2. Use it to augment the flow as much as possible.



Eventually, there are no more augmenting paths.

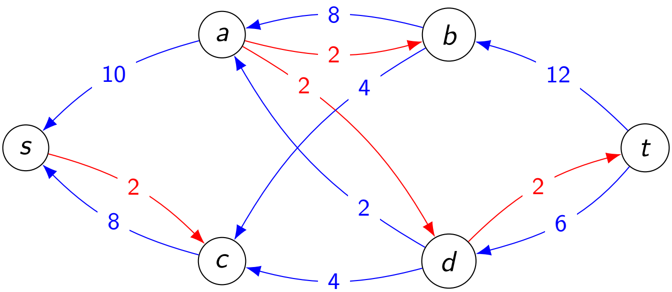
The final residual graph

We can see this in the residual graph for the final flow obtained:



The final residual graph

We can see this in the residual graph for the final flow obtained:



From s, we can only get to c. From c, we can't go anywhere new and can only return to s. **There is no s, t-path in the residual graph.**

The residual graph theorem

Theorem

Suppose that we have a network (N, A) and a feasible flow $\mathbf{x}$ such that there is no s, t-path in the residual graph. Then:

The residual graph theorem

Theorem

Suppose that we have a network (N, A) and a feasible flow $\mathbf{x}$ such that there is no s, t-path in the residual graph. Then:

Let S be the set of all nodes reachable from s in the residual graph. Let T be the set of all other nodes. The cut (S, T) has the same capacity as the value of $\mathbf{x}$.

The residual graph theorem

Theorem

Suppose that we have a network (N, A) and a feasible flow $\mathbf{x}$ such that there is no s, t-path in the residual graph. Then:

Let S be the set of all nodes reachable from s in the residual graph. Let T be the set of all other nodes. The cut (S, T) has the same capacity as the value of $\mathbf{x}$.

In particular, $\mathbf{x}$ is a maximum flow and (S, T) is a minimum cut.

The residual graph theorem

Theorem

Suppose that we have a network (N, A) and a feasible flow $\mathbf{x}$ such that there is no s, t-path in the residual graph. Then:

Let S be the set of all nodes reachable from s in the residual graph. Let T be the set of all other nodes. The cut (S, T) has the same capacity as the value of $\mathbf{x}$.

In particular, $\mathbf{x}$ is a maximum flow and (S, T) is a minimum cut.

In our example, we take $S = \{s, c\}$ and $T = \{a, b, d, t\}$. The capacity of this cut is $c_{sa} + c_{cb} + c_{cd} = 10 + 4 + 4 = 18$, same as the value of $\mathbf{x}$.

Applying the definition

In the cut (S, T) defined in the residual graph theorem, **the residual graph has no arcs from S to T.**

Applying the definition

In the cut (S, T) defined in the residual graph theorem, **the residual graph has no arcs from S to T**. What does that mean?

Applying the definition

In the cut (S, T) defined in the residual graph theorem, **the residual graph has no arcs from S to T.** What does that mean?

Recall:

- Whenever $x_{ij} < c_{ij}$ for an arc $(i, j) \in A$, the residual graph has an arc $i \to j$.
- Whenever $x_{ij} > 0$ for an arc $(i, j) \in A$, the residual graph has an arc $j \to i$.

Applying the definition

In the cut (S, T) defined in the residual graph theorem, **the residual graph has no arcs from S to T.** What does that mean?

Recall:

- Whenever $x_{ij} < c_{ij}$ for an arc $(i, j) \in A$, the residual graph has an arc $i \to j$.
- Whenever $x_{ij} > 0$ for an arc $(i, j) \in A$, the residual graph has an arc $j \to i$.

Therefore:

- For every arc (i, j) with $i \in S$ and $j \in T$, $x_{ij} = c_{ij}$.

Applying the definition

In the cut (S, T) defined in the residual graph theorem, **the residual graph has no arcs from S to T**. What does that mean?

Recall:

- Whenever $x_{ij} < c_{ij}$ for an arc $(i, j) \in A$, the residual graph has an arc $i \to j$.
- Whenever $x_{ij} > 0$ for an arc $(i, j) \in A$, the residual graph has an arc $j \to i$.

Therefore:

- For every arc (i, j) with $i \in S$ and $j \in T$, $x_{ij} = c_{ij}$.
- For every arc (i, j) with $i \in T$ and $j \in S$, $x_{ij} = 0$.

Another equation for the value

Lemma

For **any** *cut* (S, T), $v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in T} \sum_{j \in S} x_{ij}$.

(We proved this at the end of Lecture 23.)

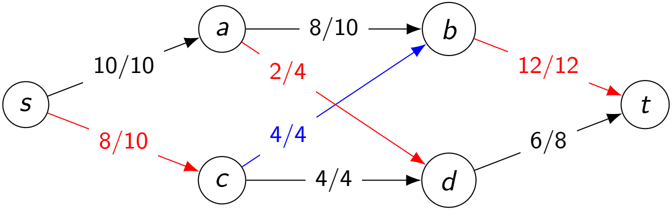
Another equation for the value

Lemma

For **any** *cut* (S, T), $v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in T} \sum_{j \in S} x_{ij}$.

(We proved this at the end of Lecture 23.)

Example: $S = \{s, a, b\}$ and $T = \{c, d, t\}$.



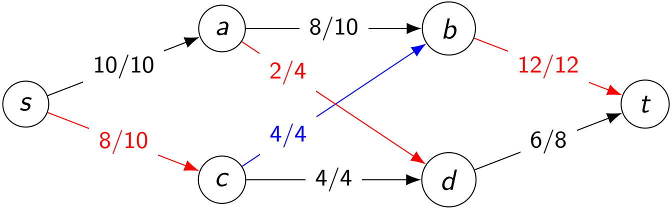
Another equation for the value

Lemma

For **any** *cut* (S, T), $v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in T} \sum_{j \in S} x_{ij}$.

(We proved this at the end of Lecture 23.)

Example: $S = \{s, a, b\}$ and $T = \{c, d, t\}$.



$$18 = v(\mathbf{x}) = x_{sc} + x_{ad} + x_{bt} - x_{cb} = 8 + 2 + 12 - 4.$$

Putting these together

If (S, T) is the cut from the residual graph, we still have

$$v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in T} \sum_{j \in S} x_{ij}.$$

Putting these together

If (S, T) is the cut from the residual graph, we still have

$$v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in T} \sum_{j \in S} x_{ij}.$$

But when $i \in S, j \in T$, we know that $x_{ij} = c_{ij}$; when $i \in T$ and $j \in S$, we know that $x_{ij} = 0$.

Putting these together

If (S, T) is the cut from the residual graph, we still have

$$v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in T} \sum_{j \in S} x_{ij}.$$

But when $i \in S, j \in T$, we know that $x_{ij} = c_{ij}$; when $i \in T$ and $j \in S$, we know that $x_{ij} = 0$. Therefore

$$v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} c_{ij} - \sum_{i \in T} \sum_{j \in S} 0$$

Putting these together

If (S, T) is the cut from the residual graph, we still have

$$v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in T} \sum_{j \in S} x_{ij}.$$

But when $i \in S, j \in T$, we know that $x_{ij} = c_{ij}$; when $i \in T$ and $j \in S$, we know that $x_{ij} = 0$. Therefore

$$v(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} c_{ij} - \sum_{i \in T} \sum_{j \in S} 0 = c(S, T).$$

This proves the residual graph theorem.

The Ford–Fulkerson algorithm

This gives us a kind of algorithm for maximum flow in a network (N, A), called the Ford–Fulkerson algorithm.

The Ford–Fulkerson algorithm

This gives us a kind of algorithm for maximum flow in a network (N, A), called the Ford–Fulkerson algorithm.

1. Begin with the zero flow: $x_{ij} = 0$ for all $(i, j) \in A$.

The Ford–Fulkerson algorithm

This gives us a kind of algorithm for maximum flow in a network (N, A), called the Ford–Fulkerson algorithm.

1. Begin with the zero flow: $x_{ij} = 0$ for all $(i, j) \in A$.
2. Repeat as long as it's possible:
   - Find an augmenting path by looking for an s, t-path in the residual graph.
   - Use it to augment the flow $\mathbf{x}$ as much as possible.

The Ford–Fulkerson algorithm

This gives us a kind of algorithm for maximum flow in a network (N, A), called the Ford–Fulkerson algorithm.

1. Begin with the zero flow: $x_{ij} = 0$ for all $(i, j) \in A$.
2. Repeat as long as it's possible:
   - Find an augmenting path by looking for an s, t-path in the residual graph.
   - Use it to augment the flow $\mathbf{x}$ as much as possible.
3. At the end, $\mathbf{x}$ is the max flow, and we can prove it: the theorem gives a cut (S, T) with $v(\mathbf{x}) = c(S, T)$.

The Ford–Fulkerson algorithm

This gives us a kind of algorithm for maximum flow in a network (N, A), called the Ford–Fulkerson algorithm.

1. Begin with the zero flow: $x_{ij} = 0$ for all $(i, j) \in A$.
2. Repeat as long as it's possible:
   - Find an augmenting path by looking for an s, t-path in the residual graph.
   - Use it to augment the flow $\mathbf{x}$ as much as possible.
3. At the end, $\mathbf{x}$ is the max flow, and we can prove it: the theorem gives a cut (S, T) with $v(\mathbf{x}) = c(S, T)$.

One lingering doubt. . .

The Ford–Fulkerson algorithm

This gives us a kind of algorithm for maximum flow in a network (N, A), called the Ford–Fulkerson algorithm.

1. Begin with the zero flow: $x_{ij} = 0$ for all $(i, j) \in A$.
2. Repeat as long as it's possible:
   - Find an augmenting path by looking for an s, t-path in the residual graph.
   - Use it to augment the flow $\mathbf{x}$ as much as possible.
3. At the end, $\mathbf{x}$ is the max flow, and we can prove it: the theorem gives a cut (S, T) with $v(\mathbf{x}) = c(S, T)$.

One lingering doubt... how do we know that the algorithm will eventually stop?

Bounds on stopping time

We can prove one (really bad) upper bound!

Bounds on stopping time

We can prove one (really bad) upper bound!

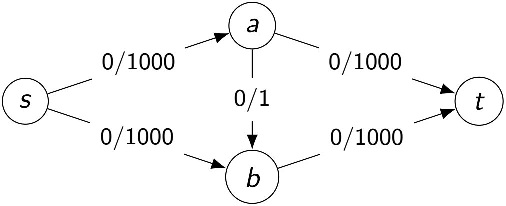
Suppose all capacities are integers. Then the value of $\mathbf{x}$ goes up by at least 1 at each step. Since $v(\mathbf{x}) \leq \sum_{j:(s,j)\in A} c_{sj}$, the algorithm must eventually stop.

Bounds on stopping time

We can prove one (really bad) upper bound!

Suppose all capacities are integers. Then the value of $\mathbf{x}$ goes up by at least 1 at each step. Since $v(\mathbf{x}) \leq \sum_{j:(s,j)\in A} c_{sj}$, the algorithm must eventually stop.

This can actually happen, if we're really bad at choosing augmenting paths:

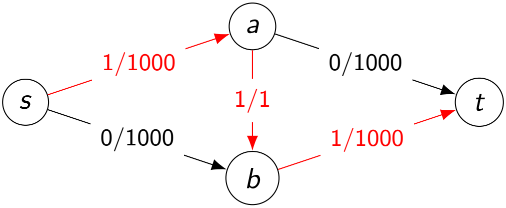


Bounds on stopping time

We can prove one (really bad) upper bound!

Suppose all capacities are integers. Then the value of $\mathbf{x}$ goes up by at least 1 at each step. Since $v(\mathbf{x}) \leq \sum_{j:(s,j)\in A} c_{sj}$, the algorithm must eventually stop.

This can actually happen, if we're really bad at choosing augmenting paths:

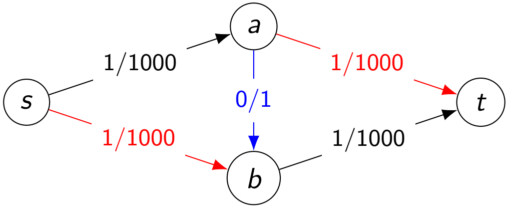


Bounds on stopping time

We can prove one (really bad) upper bound!

Suppose all capacities are integers. Then the value of $\mathbf{x}$ goes up by at least 1 at each step. Since $v(\mathbf{x}) \leq \sum_{j:(s,j)\in A} c_{sj}$, the algorithm must eventually stop.

This can actually happen, if we're really bad at choosing augmenting paths:

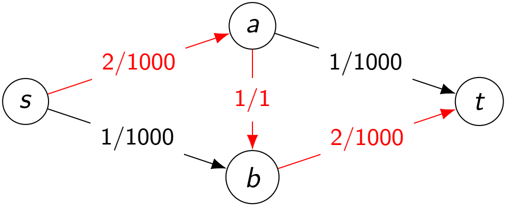


Bounds on stopping time

We can prove one (really bad) upper bound!

Suppose all capacities are integers. Then the value of $\mathbf{x}$ goes up by at least 1 at each step. Since $v(\mathbf{x}) \leq \sum_{j:(s,j)\in A} c_{sj}$, the algorithm must eventually stop.

This can actually happen, if we're really bad at choosing augmenting paths:

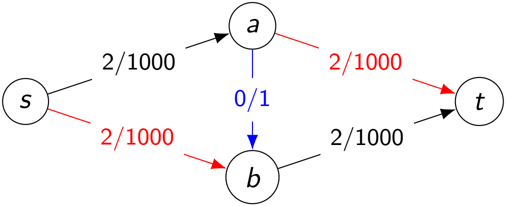


Bounds on stopping time

We can prove one (really bad) upper bound!

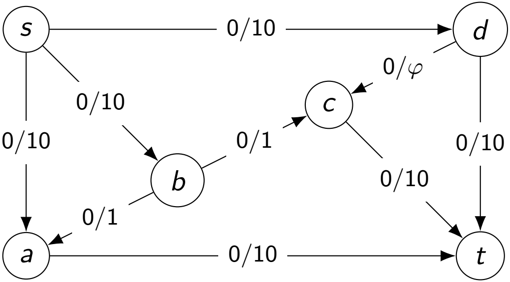
Suppose all capacities are integers. Then the value of $\mathbf{x}$ goes up by at least 1 at each step. Since $v(\mathbf{x}) \leq \sum_{j:(s,j)\in A} c_{sj}$, the algorithm must eventually stop.

This can actually happen, if we're really bad at choosing augmenting paths:



Infinite loop example

In general, if we pick our augmenting paths really badly, there are no guarantees. Example (see lecture notes for details):



One irrational capacity: $c_{dc} = \phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

The max value of 21 can be reached in 3 steps: augment along $s \to a \to t$, $s \to d \to t$, and $s \to b \to c \to t$. But it's possible to do infinitely many steps and be stuck at a value below 5.

Better guarantees and better algorithms

Suppose our network has n nodes and m arcs. (Note: $m < n^2$.)

- (Edmonds–Karp, 1972) Choose **the shortest augmenting path** at every step. Then at most nm augmenting steps are necessary: $O(nm^2)$ running time.

Better guarantees and better algorithms

Suppose our network has n nodes and m arcs. (Note: $m < n^2$.)

- (Edmonds–Karp, 1972) Choose **the shortest augmenting path** at every step. Then at most nm augmenting steps are necessary: $O(nm^2)$ running time.
- (Dinic, 1970) With further cleverness: $O(n^2 m)$ running time.

Better guarantees and better algorithms

Suppose our network has n nodes and m arcs. (Note: $m < n^2$.)

- (Edmonds–Karp, 1972) Choose **the shortest augmenting path** at every step. Then at most nm augmenting steps are necessary: $O(nm^2)$ running time.
- (Dinic, 1970) With further cleverness: $O(n^2 m)$ running time.
- (Goldberg–Tarjan, 1986) Push-relabel algorithm: also $O(n^2 m)$, but can be done more carefully in $O(n^3)$ or $O(nm \log \frac{n^2}{m})$ time.

  (See last semester's notes if you're curious.)

Better guarantees and better algorithms

Suppose our network has n nodes and m arcs. (Note: $m < n^2$.)

- (Edmonds–Karp, 1972) Choose **the shortest augmenting path** at every step. Then at most nm augmenting steps are necessary: $O(nm^2)$ running time.
- (Dinic, 1970) With further cleverness: $O(n^2 m)$ running time.
- (Goldberg–Tarjan, 1986) Push-relabel algorithm: also $O(n^2 m)$, but can be done more carefully in $O(n^3)$ or $O(nm \log \frac{n^2}{m})$ time.

  (See last semester's notes if you're curious.)
- Modern state of the art: $O(nm)$ time, by choosing between two different algorithms when m is large or small.