Algorithmic Insights III - Flows, Cuts and Transformations

K. Subramani[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

March 2, 2015

Outline

Outline

Outline

Maximum Flow

Maximum Flow

The Problem

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

1. $f(u, v) \geq 0, \forall (u, v) \in E$.

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

1. $f(u, v) \geq 0, \forall (u, v) \in E$.
2. $f(u, v) \leq c(u, v), \forall (u, v) \in E$.

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

1. $f(u, v) \geq 0, \forall (u, v) \in E$.
2. $f(u, v) \leq c(u, v), \forall (u, v) \in E$.
3. The net flow into any vertex other than s and t is

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

1. $f(u, v) \geq 0, \forall (u, v) \in E$.
2. $f(u, v) \leq c(u, v), \forall (u, v) \in E$.
3. The net flow into any vertex other than s and t is 0.

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

1. $f(u, v) \geq 0, \forall (u, v) \in E$.
2. $f(u, v) \leq c(u, v), \forall (u, v) \in E$.
3. The net flow into any vertex other than s and t is 0.

Note

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

1. $f(u, v) \geq 0, \forall (u, v) \in E$.
2. $f(u, v) \leq c(u, v), \forall (u, v) \in E$.
3. The net flow into any vertex other than s and t is 0.

Note

The value of a flow is the net flow out of s

Maximum Flow

The Problem

Given a directed graph $G = \langle V, E \rangle$ with n vertices and m edges and a capacity function $c : E \to Z$, find the value of the maximum flow from s to t.

Properties

Any flow $f : V \times V \to Z$ must satisfy the following properties:

1. $f(u, v) \geq 0, \forall (u, v) \in E$.
2. $f(u, v) \leq c(u, v), \forall (u, v) \in E$.
3. The net flow into any vertex other than s and t is 0.

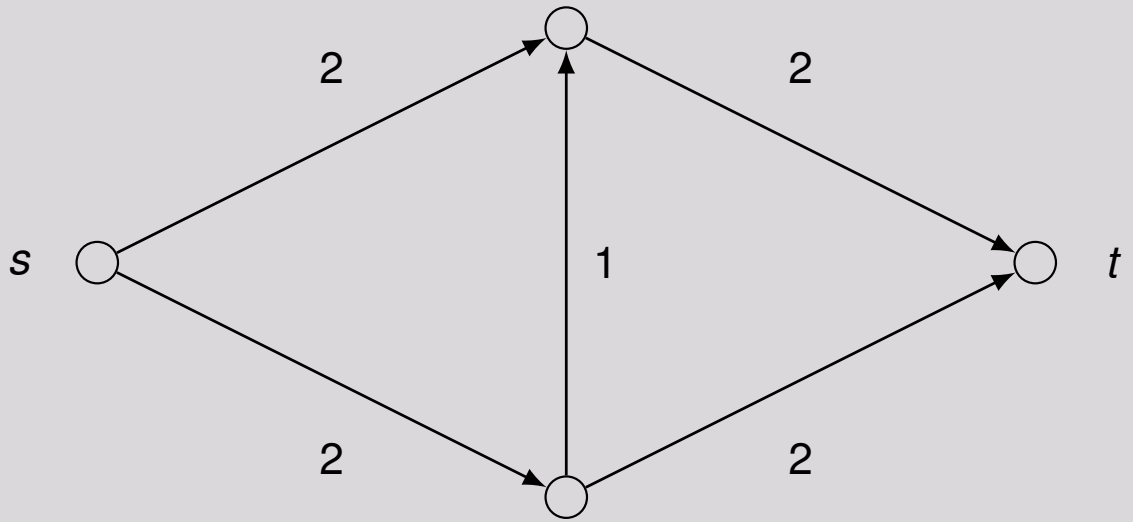
Note

The value of a flow is the net flow out of s (or the net flow into t) and it is denoted by $|f|$.

Problem example

Problem example

Example



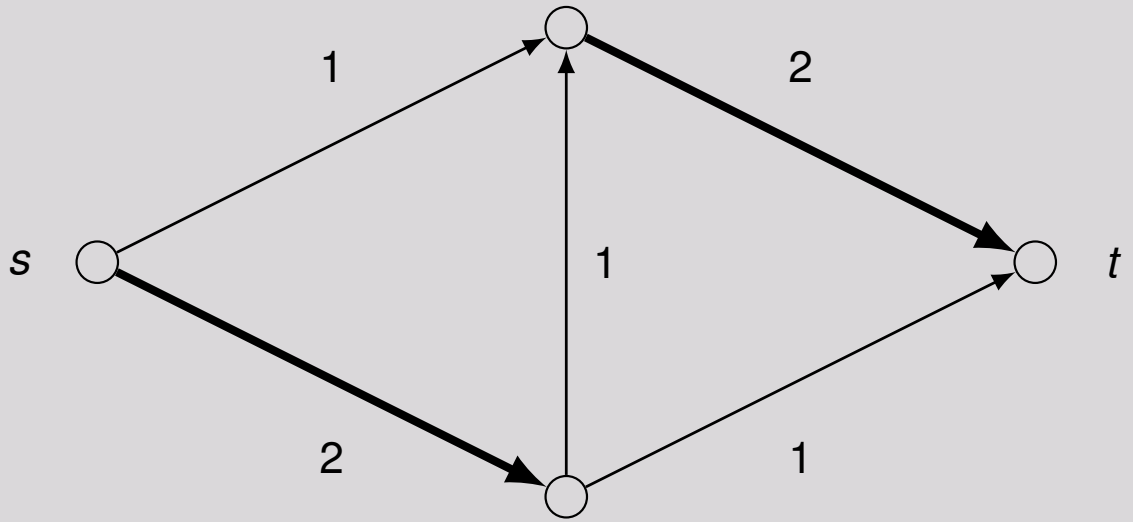
Sample flows

Sample flows

Example

Sample flows

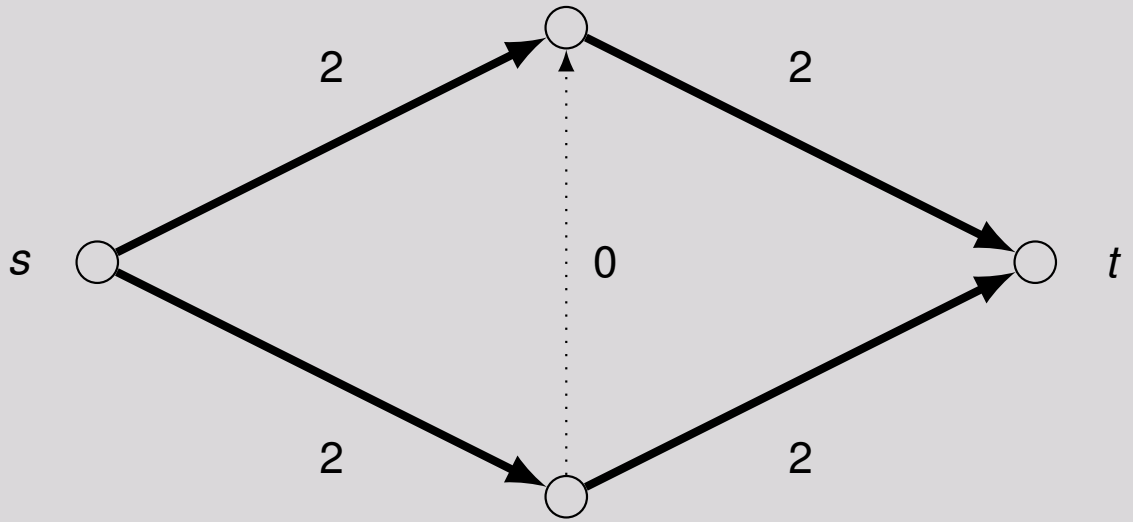
Example



Another flow

Another flow

Example



Some observations

Some observations

Observations

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph.*

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph. This permits us to reverse flow if needed.*

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph. This permits us to reverse flow if needed.*

   Indeed, there will be a reverse edge for every edge on which there is some positive flow and the capacity of that edge will be $f(e)$.

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph. This permits us to reverse flow if needed.*

   Indeed, there will be a reverse edge for every edge on which there is some positive flow and the capacity of that edge will be $f(e)$. In other words, $c_f(\bar{e}) = f(e)$.

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph. This permits us to reverse flow if needed.*

   Indeed, there will be a reverse edge for every edge on which there is some positive flow and the capacity of that edge will be $f(e)$. In other words, $c_f(\bar{e}) = f(e)$.

   The reverse edge corresponding to edge e, is denoted by $\bar{e}$.

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph. This permits us to reverse flow if needed.*

   Indeed, there will be a reverse edge for every edge on which there is some positive flow and the capacity of that edge will be $f(e)$. In other words, $c_f(\bar{e}) = f(e)$.

   The reverse edge corresponding to edge e, is denoted by $\bar{e}$.
5. *Any path from s to t in G_f is called an augmenting path.*

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph. This permits us to reverse flow if needed.*

   Indeed, there will be a reverse edge for every edge on which there is some positive flow and the capacity of that edge will be $f(e)$. In other words, $c_f(\bar{e}) = f(e)$.

   The reverse edge corresponding to edge e, is denoted by $\bar{e}$.
5. *Any path from s to t in G_f is called an augmenting path.*

Example

Some observations

Observations

1. *Flow can be increased on an edge, only if there is some residual capacity on that edge.*
2. *We can increase flow along a path from s to t, only if every edge on that path has some residual capacity.*
3. *Construct the residual network G_f, where edge has capacity $c_f(e) = c(e) - f(e)$.*
4. *Note that in the residual graph there could be edges which are not edges in the original graph. This permits us to reverse flow if needed.*

   Indeed, there will be a reverse edge for every edge on which there is some positive flow and the capacity of that edge will be $f(e)$. In other words, $c_f(\bar{e}) = f(e)$.

   The reverse edge corresponding to edge e, is denoted by $\bar{e}$.
5. *Any path from s to t in G_f is called an augmenting path.*

Example

Draw the residual graph corresponding to the flow in Figure 2.

Augmenting path theorem

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

We will prove the following equivalent statement:

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

We will prove the following equivalent statement:
A flow f is not maximal if and only of there is an augmenting path in G_f.

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

We will prove the following equivalent statement:
A flow f is not maximal if and only of there is an augmenting path in G_f.

Only If:

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

We will prove the following equivalent statement:
A flow f is not maximal if and only of there is an augmenting path in G_f.

Only If: Assume there exists an augmenting path δ in G_f.

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

We will prove the following equivalent statement:
A flow f is not maximal if and only of there is an augmenting path in G_f.

Only If: Assume there exists an augmenting path δ in G_f.

Clearly, we can increase flow by at least one unit along this path so that all the flow constraints are met.

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

We will prove the following equivalent statement:
A flow f is not maximal if and only of there is an augmenting path in G_f.

Only If: Assume there exists an augmenting path δ in G_f.

Clearly, we can increase flow by at least one unit along this path so that all the flow constraints are met.

Since the net flow out of s is increased by at least one unit, it follows that f is not optimal.

Augmenting path theorem

Theorem

A flow f is maximal if and only if there is no augmenting path in G_f.

If there is an augmenting path, increasing flow along that path, produces a flow of greater value.

Proof

We will prove the following equivalent statement:
A flow f is not maximal if and only of there is an augmenting path in G_f.

Only If: Assume there exists an augmenting path δ in G_f.

Clearly, we can increase flow by at least one unit along this path so that all the flow constraints are met.

Since the net flow out of s is increased by at least one unit, it follows that f is not optimal.

Proof of Augmenting path theorem

Proof of Augmenting path theorem

Proof (contd.)

If:

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal.

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\Delta(e) \quad =$$

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\Delta(e) \quad = \quad \max(0, f'(e) - f(e))$$

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\begin{aligned} \Delta(e) &= \max(0, f'(e) - f(e)) \\ \Delta(e') &= \end{aligned}$$

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\begin{aligned} \Delta(e) &= \max(0, f'(e) - f(e)) \\ \Delta(e') &= \max(0, f(e) - f'(e)) \end{aligned}$$

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\begin{aligned} \Delta(e) &= \max(0, f'(e) - f(e)) \\ \Delta(e') &= \max(0, f(e) - f'(e)) \end{aligned}$$

It is not hard to see that the flow Δ satisfies all the properties of a flow in G_f,

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\begin{aligned} \Delta(e) &= \max(0, f'(e) - f(e)) \\ \Delta(e') &= \max(0, f(e) - f'(e)) \end{aligned}$$

It is not hard to see that the flow Δ satisfies all the properties of a flow in G_f, i.e., it is a legal flow on G_f.

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\begin{aligned} \Delta(e) &= \max(0, f'(e) - f(e)) \\ \Delta(e') &= \max(0, f(e) - f'(e)) \end{aligned}$$

It is not hard to see that the flow Δ satisfies all the properties of a flow in G_f, i.e., it is a legal flow on G_f. Furthermore, $\Delta = |f'| - |f|$

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\begin{aligned} \Delta(e) &= \max(0, f'(e) - f(e)) \\ \Delta(e') &= \max(0, f(e) - f'(e)) \end{aligned}$$

It is not hard to see that the flow Δ satisfies all the properties of a flow in G_f, i.e., it is a legal flow on G_f. Furthermore, $\Delta = |f'| - |f| > 0$.

Proof of Augmenting path theorem

Proof (contd.)

If: Assume that f is not maximal. There exists a flow f' such that $|f'| > |f|$.

Consider the differential flow Δ defined as follows:

$$\begin{aligned} \Delta(e) &= \max(0, f'(e) - f(e)) \\ \Delta(e') &= \max(0, f(e) - f'(e)) \end{aligned}$$

It is not hard to see that the flow Δ satisfies all the properties of a flow in G_f, i.e., it is a legal flow on G_f. Furthermore, $\Delta = |f'| - |f| > 0$.

But this is possible only if there is a path composed of edges with non-zero capacity from s to t in G_f!

The Ford-Fulkerson Algorithm

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

1: Start with the everywhere 0 flow, f.

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

1: Start with the everywhere 0 flow, f.
2: Construct the residual network G_f.

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

1: Start with the everywhere 0 flow, f.
2: Construct the residual network G_f.
3: **while** (there exists a path from s to t in G_f) **do**

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

1: Start with the everywhere 0 flow, f.
2: Construct the residual network G_f.
3: **while** (there exists a path from s to t in G_f) **do**
4: Augment flow along this path by 1.

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

1: Start with the everywhere 0 flow, f.
2: Construct the residual network G_f.
3: **while** (there exists a path from s to t in G_f) **do**
4: Augment flow along this path by 1.
5: Set f to the new flow.

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

1: Start with the everywhere 0 flow, f.
2: Construct the residual network G_f.
3: **while** (there exists a path from s to t in G_f) **do**
4: Augment flow along this path by 1.
5: Set f to the new flow.
6: Compute G_f.

The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Function FORD-FULKERSON(G, c)

1: Start with the everywhere 0 flow, f.
2: Construct the residual network G_f.
3: **while** (there exists a path from s to t in G_f) **do**
4: Augment flow along this path by 1.
5: Set f to the new flow.
6: Compute G_f.
7: **end while**

Algorithm 2.9: The Ford-Fulkerson procedure

More observations

More observations

Observations

More observations

Observations

1. *The running time of the above procedure is exponential in n.*

More observations

Observations

1. *The running time of the above procedure is exponential in n.*
2. *Augmenting along the shortest path from s to t gives polynomial time convergence*

More observations

Observations

1. *The running time of the above procedure is exponential in n.*
2. *Augmenting along the shortest path from s to t gives polynomial time convergence (Edmonds-Karp algorithm).*

More observations

Observations

1. *The running time of the above procedure is exponential in n.*
2. *Augmenting along the shortest path from s to t gives polynomial time convergence (Edmonds-Karp algorithm).*
3. *Alternatively, augmenting along the path with the largest "bottleneck" capacity establishes polynomial time convergence.*

More observations

Observations

1. *The running time of the above procedure is exponential in n.*
2. *Augmenting along the shortest path from s to t gives polynomial time convergence (Edmonds-Karp algorithm).*
3. *Alternatively, augmenting along the path with the largest "bottleneck" capacity establishes polynomial time convergence.*
4. *If the capacities are integers, there is a maximal flow f, where $f(e)$ is integral for all $e \in E$.*

More observations

Observations

1. *The running time of the above procedure is exponential in n.*
2. *Augmenting along the shortest path from s to t gives polynomial time convergence (Edmonds-Karp algorithm).*
3. *Alternatively, augmenting along the path with the largest "bottleneck" capacity establishes polynomial time convergence.*
4. *If the capacities are integers, there is a maximal flow f, where $f(e)$ is integral for all $e \in E$.*
5. *If reverse edges are not allowed, then it is possible for the Ford-Fulkerson procedure to get stuck in a local minimum.*

Min Cuts

Min Cuts

Definition

Given a capacitated graph $G = \langle V, E \rangle$, with source s and destination t, a cut C is a set of edges whose removal separates s from t.

Min Cuts

Definition

Given a capacitated graph $G = \langle V, E \rangle$, with source s and destination t, a cut C is a set of edges whose removal separates s from t.

Alternatively, a cut is partitioning of the vertices of G into two disjoint sets S and T, such that $s \in S$ and $t \in T$. Then, C consists of the edges that cross from S to T.

Min Cuts

Definition

Given a capacitated graph $G = \langle V, E \rangle$, with source s and destination t, a cut C is a set of edges whose removal separates s from t.

Alternatively, a cut is partitioning of the vertices of G into two disjoint sets S and T, such that $s \in S$ and $t \in T$. Then, C consists of the edges that cross from S to T.

The weight of a cut is the sum of the weights of the edges in that cut.

Min Cuts

Definition

Given a capacitated graph $G = \langle V, E \rangle$, with source s and destination t, a cut C is a set of edges whose removal separates s from t.

Alternatively, a cut is partitioning of the vertices of G into two disjoint sets S and T, such that $s \in S$ and $t \in T$. Then, C consists of the edges that cross from S to T.

The weight of a cut is the sum of the weights of the edges in that cut.

The Problem

Min Cuts

Definition

Given a capacitated graph $G = \langle V, E \rangle$, with source s and destination t, a cut C is a set of edges whose removal separates s from t.

Alternatively, a cut is partitioning of the vertices of G into two disjoint sets S and T, such that $s \in S$ and $t \in T$. Then, C consists of the edges that cross from S to T.

The weight of a cut is the sum of the weights of the edges in that cut.

The Problem

Given a capacitated graph $G = \langle V, E \rangle$, with source s and destination t, find the weight of the minimum cut that separates s from t.

Observations

Observations

Observations

Observations

Observations

1. *The value of the any flow is at most the value of any cut.*

Observations

Observations

1. *The value of the any flow is at most the value of any cut.*
2. *Thus, the value of the maximum flow is at most the value of the minimum cut.*

Observations

Observations

1. *The value of the any flow is at most the value of any cut.*
2. *Thus, the value of the maximum flow is at most the value of the minimum cut.*
3. *Given a flow f and given a cut (S, T), the net flow across a cut is the value of that flow,*

Observations

Observations

1. *The value of the any flow is at most the value of any cut.*
2. *Thus, the value of the maximum flow is at most the value of the minimum cut.*
3. *Given a flow f and given a cut (S, T), the net flow across a cut is the value of that flow, i.e., $f(S, T) = |f|$.*

Observations

Observations

1. *The value of the any flow is at most the value of any cut.*
2. *Thus, the value of the maximum flow is at most the value of the minimum cut.*
3. *Given a flow f and given a cut (S, T), the net flow across a cut is the value of that flow, i.e., $f(S, T) = |f|$.*
4. *With some thought, it should be clear that the value of the maximum flow is equal the value of the minimum cut*

Observations

Observations

1. *The value of the any flow is at most the value of any cut.*
2. *Thus, the value of the maximum flow is at most the value of the minimum cut.*
3. *Given a flow f and given a cut (S, T), the net flow across a cut is the value of that flow, i.e., $f(S, T) = |f|$.*
4. *With some thought, it should be clear that the value of the maximum flow is equal the value of the minimum cut (Max-Flow Min cut theorem)*

Observations

Observations

1. *The value of the any flow is at most the value of any cut.*
2. *Thus, the value of the maximum flow is at most the value of the minimum cut.*
3. *Given a flow f and given a cut (S, T), the net flow across a cut is the value of that flow, i.e., $f(S, T) = |f|$.*
4. *With some thought, it should be clear that the value of the maximum flow is equal the value of the minimum cut (Max-Flow Min cut theorem)*
5. *The Max Flow and Min Cut problems are thus dual to each other.*

Max-Flow Min-Cut Theorem

Max-Flow Min-Cut Theorem

Proof

Max-Flow Min-Cut Theorem

Proof

1. Let S denote the set of vertices reachable from s in G_f.

Max-Flow Min-Cut Theorem

Proof

1. Let S denote the set of vertices reachable from s in G_f. Let T denote the rest of the vertices, including t.

Max-Flow Min-Cut Theorem

Proof

1. Let S denote the set of vertices reachable from s in G_f. Let T denote the rest of the vertices, including t.
2. Consider an edge e, crossing from s to t.

Max-Flow Min-Cut Theorem

Proof

1. Let S denote the set of vertices reachable from s in G_f. Let T denote the rest of the vertices, including t.
2. Consider an edge e, crossing from s to t. It is clear that $c_f(e) = 0$. It follows that $f(e) = c(e)$.

Max-Flow Min-Cut Theorem

Proof

1. Let S denote the set of vertices reachable from s in G_f. Let T denote the rest of the vertices, including t.
2. Consider an edge e, crossing from s to t. It is clear that $c_f(e) = 0$. It follows that $f(e) = c(e)$.
3. Each cut edge is *saturated* by f.

Max-Flow Min-Cut Theorem

Proof

1. Let S denote the set of vertices reachable from s in G_f. Let T denote the rest of the vertices, including t.
2. Consider an edge e, crossing from s to t. It is clear that $c_f(e) = 0$. It follows that $f(e) = c(e)$.
3. Each cut edge is *saturated* by f.
4. The weight of the cut (S, T) is equal to the sum of the weights of the saturated edges, which is clearly $|f|$.

Max-Flow Min-Cut Theorem

Proof

1. Let S denote the set of vertices reachable from s in G_f. Let T denote the rest of the vertices, including t.
2. Consider an edge e, crossing from s to t. It is clear that $c_f(e) = 0$. It follows that $f(e) = c(e)$.
3. Each cut edge is *saturated* by f.
4. The weight of the cut (S, T) is equal to the sum of the weights of the saturated edges, which is clearly $|f|$.
5. The above theorem can also be proved using linear programming duality.

Notion of reduction

Notion of reduction

Definition

A reduction from problem A to problem B,

Notion of reduction

Definition

A reduction from problem A to problem B, is a transformation function f that maps "yes"-instances of A into "yes"-instances of B,

Notion of reduction

Definition

A reduction from problem A to problem B, is a transformation function f that maps "yes"-instances of A into "yes"-instances of B, and "no"-instances of A into "no"-instances of B.

Notion of reduction

Definition

A reduction from problem A to problem B, is a transformation function f that maps "yes"-instances of A into "yes"-instances of B, and "no"-instances of A into "no"-instances of B.

Definition

A reduction is said to be polynomial time

Notion of reduction

Definition

A reduction from problem A to problem B, is a transformation function f that maps "yes"-instances of A into "yes"-instances of B, and "no"-instances of A into "no"-instances of B.

Definition

A reduction is said to be polynomial time (or log space) the transformation function $f()$ can be computed in polynomial time

Notion of reduction

Definition

A reduction from problem A to problem B, is a transformation function f that maps "yes"-instances of A into "yes"-instances of B, and "no"-instances of A into "no"-instances of B.

Definition

A reduction is said to be polynomial time (or log space) the transformation function $f()$ can be computed in polynomial time (or log space).

Important Observations

Important Observations

Observations

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A,*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*
4. *B generalizes A*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*
4. *B generalizes A or A is a special case of B.*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*
4. *B generalizes A or A is a special case of B.*
5. *If B is in* **P**, *then so is A.*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*
4. *B generalizes A or A is a special case of B.*
5. *If B is in* **P**, *then so is A.*
6. *If A is not in* **P**, *then so is B.*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*
4. *B generalizes A or A is a special case of B.*
5. *If B is in* **P**, *then so is A.*
6. *If A is not in* **P**, *then so is B. (Reduction in* **NP**).

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*
4. *B generalizes A or A is a special case of B.*
5. *If B is in* **P**, *then so is A.*
6. *If A is not in* **P**, *then so is B. (Reduction in* **NP**).
7. *The function $f()$ needs to be circumscribed severely if conclusions are to be meaningful.*

Important Observations

Observations

1. *We write $A \leq_f B$ to denote that A can be reduced to B through $f()$.*
2. *We can solve A, If we know how to solve B, using B as a sub-routine.*
3. *B is at least as hard as A.*
4. *B generalizes A or A is a special case of B.*
5. *If B is in* **P**, *then so is A.*
6. *If A is not in* **P**, *then so is B. (Reduction in* **NP**).
7. *The function $f()$ needs to be circumscribed severely if conclusions are to be meaningful. (Reduce TSP to graph reachability).*

Simple Reductions

Simple Reductions

Sorting and Find-Min

Simple Reductions

Sorting and Find-Min

1. Reduce the Find-Min problem to sorting.

Simple Reductions

Sorting and Find-Min

1. Reduce the Find-Min problem to sorting.
2. Reduce the Sorting problem to FInd-Min.

The perfect matching problem

The perfect matching problem

Definition

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$,

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$, a matching M is any collection of vertex-disjoint edges.

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$, a matching M is any collection of vertex-disjoint edges.

The matching is said to be perfect, if $|M| = \frac{n}{2}$.

The Problem

Given a bipartite graph $G = \langle L, R, E \rangle$,

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$, a matching M is any collection of vertex-disjoint edges.

The matching is said to be perfect, if $|M| = \frac{n}{2}$.

The Problem

Given a bipartite graph $G = \langle L, R, E \rangle$, does G have a perfect matching?

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$, a matching M is any collection of vertex-disjoint edges.

The matching is said to be perfect, if $|M| = \frac{n}{2}$.

The Problem

Given a bipartite graph $G = \langle L, R, E \rangle$, does G have a perfect matching? We can assume that $|L| = |R|$, or else the problem is trivial.

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$, a matching M is any collection of vertex-disjoint edges.

The matching is said to be perfect, if $|M| = \frac{n}{2}$.

The Problem

Given a bipartite graph $G = \langle L, R, E \rangle$, does G have a perfect matching? We can assume that $|L| = |R|$, or else the problem is trivial.

The following problem generalizes the Perfect Matching problem:

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$, a matching M is any collection of vertex-disjoint edges.

The matching is said to be perfect, if $|M| = \frac{n}{2}$.

The Problem

Given a bipartite graph $G = \langle L, R, E \rangle$, does G have a perfect matching? We can

assume that $|L| = |R|$, or else the problem is trivial.

The following problem generalizes the Perfect Matching problem:
Given a bipartite graph $G = \langle L, R, E \rangle$,

The perfect matching problem

Definition

Given an undirected graph $G = \langle V, E \rangle$, a matching M is any collection of vertex-disjoint edges.

The matching is said to be perfect, if $|M| = \frac{n}{2}$.

The Problem

Given a bipartite graph $G = \langle L, R, E \rangle$, does G have a perfect matching? We can

assume that $|L| = |R|$, or else the problem is trivial.

The following problem generalizes the Perfect Matching problem:
Given a bipartite graph $G = \langle L, R, E \rangle$, what is the maximum matching in G?

The transformation of bipartite matching to Max-Flow

The transformation of bipartite matching to Max-Flow

Reduction

The transformation of bipartite matching to Max-Flow

Reduction

1. Create a new vertex s and draw an arc from s to all the vertices in L.

The transformation of bipartite matching to Max-Flow

Reduction

1. Create a new vertex s and draw an arc from s to all the vertices in L.
2. Create a new vertex t and draw an arch from each vertex in R to t.

The transformation of bipartite matching to Max-Flow

Reduction

1. Create a new vertex s and draw an arc from s to all the vertices in L.
2. Create a new vertex t and draw an arch from each vertex in R to t.
3. All edges have capacity 1.

The transformation of bipartite matching to Max-Flow

Reduction

1. Create a new vertex s and draw an arc from s to all the vertices in L.
2. Create a new vertex t and draw an arch from each vertex in R to t.
3. All edges have capacity 1.
4. G has a perfect matching if and only if the maximum flow from s to t in the transformed graph is $|L|$.

The transformation of bipartite matching to Max-Flow

Reduction

1. Create a new vertex s and draw an arc from s to all the vertices in L.
2. Create a new vertex t and draw an arch from each vertex in R to t.
3. All edges have capacity 1.
4. G has a perfect matching if and only if the maximum flow from s to t in the transformed graph is $|L|$.
5. More generally,

$$\text{Max-Bipartite Matching} \leq \text{Max-Flow}$$