Max-Flow Problems

Max-Flow is a graph problem that

- seems very specific and narrowly defined.
- But many seemingly unrelated problems can be converted to max-flow problems.

A flow network consists of:

- A directed graph $G = (V, E)$.
- Each edge $u \to v \in E$ has a capacity $c(u, v) \geq 0$. (If $u \to v \notin E$ then $c(u, v) = 0$.)
- Two special vertices: the source s and the sink t.
- For each vertex $v \in V$, there is a directed path $s \to t$ passing through v.

Note: The last condition is not essential. It is included here for convenience.

Max-Flow: Definitions

Flow Function

A flow is a real valued function $f : V \times V \rightarrow R$ that satisfies the following conditions:

- Capacity Constraint: For all $u, v \in V$, $f(u, v) \leq c(u, v)$.
- Skew Symmetry Constraint: For all $u, v \in V$, $f(v, u) = -f(u, v)$.
- Flow Conservation Constraint: For any $u \in V - \{s, t\}$,

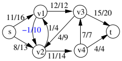
$$\sum_{v \in V} f(u, v) = 0$$

- The flow value of f is defined to be:

$$|f| = \sum_{v \in V} f(s, v)$$

$f(u, v)$ is called the net flow from u to v.

Max-Flow: Example



- In this figure, the notation $11/16$ means $f(s, v_1) = 11$ and $c(s, v_1) = 16$.
- The edges with 0 capacity are not shown.
- Only positive flow values are shown. (Recall that $f(v, u) = -f(u, v)$ by the skew symmetry constraint.)
- The capacity constraint is satisfied at all edges.
- The conservation constraint at the vertex v_1 is:

$$\begin{array}{lllllllllll} & f(v_1, s) & + & f(v_1, v_2) & + & f(v_1, v_3) & + & f(v_1, v_4) & + & f(v_1, t) & \\ = & -11 & + & -1 & + & 12 & + & 0 & + & 0 & = 0 \end{array}$$

- The flow value is: $|f| = 11 + 8 = 19$.

Max-Flow: Definition

Caution

- In the example above, suppose that 3 units flow $v_2 \to v_1$, and 2 units flow $v_1 \to v_2$.
- It can be seen that the flow conservation and the capacity constrains are still satisfied.
- But are $f(v_2 \to v_1) = 3$ and $f(v_1 \to v_2) = 2$? Then the Skew Symmetry constrains $f(v_1, v_2) = -f(v_2, v_1)$ would not be satisfied.
- In the formulation of the max-flow problem, shipping 3 units flow from v_2 to v_1, and 2 units flow from v_1 to v_2 is equivalent to shipping 1 unit flow from v_2 to v_1, and nothing from v_1 to v_2. In other words, the 2 units flow from v_2 to v_1, then from v_1 back to v_2 are canceled.
- So we have: $f(v_2, v_1) = 1$ and $f(v_1, v_2) = -1$

Max-Flow: Application

- $G = (V, E)$ represents an oil pipeline system
- Each edge $u \to v \in E$ is an one-directional pipeline.
- Each vertex is a site where the pipelines meet and the oil flow can be redistributed.
- $c(u, v)$ is the capacity of the pipeline $u \to v$. (Namely, the amount of oils that can flow through the pipeline per unit time.)
- s is the source of the oil, and t is the destination of the oil.
- $f(u, v)$ is the amount of oil flow through the line $u \to v$.
- Capacity Constraint: The amount of oil flow through a line cannot be more than its capacity.
- Skew Symmetry Constraint: The amount of oil flow from u to v is the negative of the amount of oil flow from v to u.

Max-Flow: Application

- Conservation Constraint: At any site $u \neq s, t$, the total amount of oil that flow into and out of u must be the same. (Namely, oil cannot be produced nor consumed at u.)
- The flow value $|f|$ is the total amount of oil flow out of s.
- We will show $|f|$ is also the total amount of oil flow into t.
- In other words, $|f|$ is the total amount of oil that flow through the pipeline system from s to t.

Max-Flow: Application

- $G = (V, E)$ represents an internet communication network.
- Each edge $u \to v \in E$ is an one-directional communication link.
- Each vertex is a site where the links meet and the data traffic can be redistributed.
- $c(u, v)$ is the bandwidth of the link $u \to v$. (Namely, $c(u, v) =$ the number of G bytes the link $u \to v$ can transmit per second.)
- s is the source of the data flow, and t is the destination of the data flow.
- $f(u, v)$ is the amount of data flow through the link $u \to v$.
- Capacity, Skew Symmetry, and Flow Conservation Constraints can be interpreted as before.
- $|f|$ is the total bandwidth of data flow from s to t through the network.

Max-Flow: Definition

Max-Flow Problem

Input: A flow network $G = (V, E)$, source s, sink t, capacity function $c(*)$
Find: A flow function $f(*)$ so that $|f|$ is maximum.

- This is the basic max-flow problem.
- There are other versions of similar problems.
- Some can be easily converted to the basic problem.
- Some cannot. But they can be solved by using similar ideas for solving the basic problem.
- We discuss a few examples.

Max-Flow: Multiple Sources and Multiple Sinks

Multiple Sources and Multiple Sinks

It is the same as the basic problems, except that:

- There are several sources $s_1, s_2, \ldots, s_p$ and several sinks $t_1, t_2, \ldots, t_q$.
- The flow conservation constrain must be satisfied at any vertex $u \in V - \{s_1, s_2, \ldots, s_p, t_1, t_2, \ldots, t_q\}$.
- The flow value $|f|$ is the sum of total flow out of all sources.

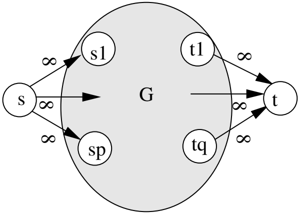
Applications

- In the oil pipeline application, there are more than one oil source and more than one destination. (Here we assume the oil from different sources are identical.)

Max-Flow: Multiple Sources and Multiple Sinks

This problem can be easily converted to the basic problem:

- Add a new source s and a new sink t.
- Add new edges $s \rightarrow s_i$ for all $1 \leq i \leq p$ and new edges $t_j \rightarrow t$ for all $1 \leq j \leq q$.
- Set the capacity of these new edges to be ∞.



The max flow function of the converted network gives the answer of the original problem.

Max-Flow: Undirected Edges

Undirected Edges

It is the same as the basic problem, except that the edges are undirected.

Applications

- In the oil pipeline application, oil can flow in either direction in pipelines.
- In the internet communication application, data can be transmitted in either direction.

This problem can be easily converted to the basic problem: For each undirected edge $e = (u, v)$, replace e by a pair of directed edges $u \to v$ and $v \to u$. The capacity of these two edges are the same as the capacity of e.

Max-Flow: Vertex Capacity

Vertex Capacity

It is the same as the basic problem, except that:

- Each vertex $u \in V - \{s, t\}$ has a capacity $c(u) \geq 0$.
- The total amount of flow going through u is at most $c(u)$. Namely:

$$\sum_{v \in V \text{ and } f(u,v) > 0} f(u, v) \leq c(u)$$

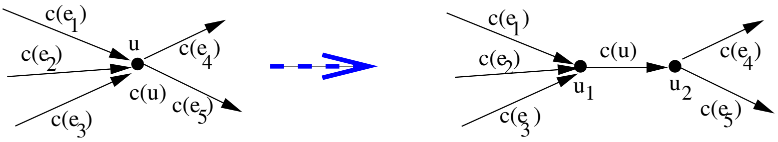
Application

In the oil pipeline application, each vertex u is an oil pumping station. $c(u)$ is the capacity of the pump.

Max-Flow: Vertex Capacity

This problem can be easily converted to the basic problem.

- For each vertex $u \neq s, t$, replace u by two new vertices u_1 and u_2.
- Add a new edge $u_1 \to u_2$, with capacity $c(u_1 \to u_2) = c(u)$.
- For each edge $v \to u$, replace it by $v \to u_1$ with the same capacity.
- For each edge $u \to w$, replace it by $u_2 \to w$ with the same capacity.



Max-Flow: Min-Cost Max-Flow Flow

Max-Flow: With Flow Cost

It is the same as the basic problem, except that:

- For each edge $u \to v$, in addition to capacity, there is a cost$(u \to v) \geq 0$. (Meaning: you must pay cost$(u \to v)$ in order to ship 1 unit flow through the edge $u \to v$.)
- The goal of the problem is to find a flow function $f(*)$ such that:
  - The flow value $|f|$ is maximum.
  - The total cost of f

$$\text{cost}(f) = \sum_{e=u \to v \in E} f(u \to v) \cdot \text{cost}(u \to v)$$

  is minimum.

- This problem cannot be easily converted to the basic problem.
- It can be solved by using similar idea.

Max-Flow: Multi-commodity Flow Problem

- We have a directed graph G. Each edge e has a capacity $c(e) \geq 0$.
- We are given t commodities $K_1, K_2, \ldots, K_t$.
- Each K_i is a triple (s_i, t_i, d_i): s_i is the source of the commodity, t_i is the destination of the commodity, d_i is the demand of the commodity.
- Interpretation: we need to ship d_i units of commodity i from s_i to t_i.
- We define a flow for commodity i: $f_i : V \times V \to R$. $f_i(*)$ must satisfy the Capacity, Skew Symmetry and Flow Conservation constraints.
- In addition, we require Aggregate Capacity constraint: For any edge $e = u \to v \in E$:

$$f(u,v) = \sum_{i=1}^{t} f_i(u,v) \leq c(u,v)$$

Max-Flow: Multi-commodity Flow Problem

Question:

Can we find flow functions $f_1, f_2, \ldots, f_t$ that satisfy all of these constraints, and also satisfy the demands for all commodities?

Application

In the Internet connetion network example, we must transmit d_i units data from the site s_i to the site t_i.

This problem cannot be converted to the basic max-flow problems, and is significantly harder to solve.

Max-Flow: Properties

Definition

Let $X \subseteq V$ and $Y \subseteq V$ be two subsets of V. The total flow from X to Y is:

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x \to y)$$

- When $X = \{u\}$ is a single vertex set, we write $f(u, Y)$ instead of $f(\{u\}, Y)$.
- By the definition: $|f| = \sum_{v \in V} f(s, v) = f(s, V)$.
- The Flow Conservation Constraint is: For any $u \neq s, t$ we must have $f(u, V) = 0$.

Max-Flow: Properties

Lemma

Let $G=(V,E)$ be a flow network and f a flow function of G. Let X,Y and Z be any subsets of vertices of G. Then:

1. $f(X,X)=0$.
2. $f(X,Y)=-f(Y,X)$.
3. If $X\cap Y=\emptyset$, then $f(X\cup Y,Z)=f(X,Z)+f(Y,Z)$.
4. If $X\cap Y=\emptyset$, then $f(Z,X\cup Y)=f(Z,X)+f(Z,Y)$.

- Statement 1: the total flow from a subset into itself is 0.
- Statement 2: the total flow from X into Y is equal to the negative of the total flow from Y into X.
- Statement 3: If $X$and Y are disjoint, then the total flow from $X\cup Y$ to Z is the sum of the flow from X to Z and the flow from Y to Z.

Max-Flow: Properties

Proof.

2.

$$\begin{array}{rcl} f(X,Y) & = & \sum_{x\in X}\sum_{y\in Y} f(x,y) \quad \text{(by skew symmetry property)} \\ & = & \sum_{x\in X}\sum_{y\in Y} -f(y,x) = -\sum_{y\in Y}\sum_{x\in X} f(y,x) \\ & = & -f(Y,X) \quad \text{(by the definition of } f(Y,X).\text{)} \end{array}$$

1. Since statement 2 is true for any X and Y, plug in $Y = X$: $f(X,X) = -f(X,X)$. This implies $f(X,X) = 0$.

3.

$$\begin{array}{rcl} f(X\cup Y, Z) & = & \sum_{x\in X\cup Y}\sum_{z\in Z} f(x,z) \quad \text{(because } X\cap Y = \emptyset\text{)} \\ & = & \sum_{x\in X}\sum_{z\in Z} f(x,z) + \sum_{x\in Y}\sum_{z\in Z} f(x,z) \\ & = & f(X,Z) + f(Y,Z) \end{array}$$

4. Similar to 3.

□

Max-Flow: Properties

Fact

We can show $|f| = f(V,t)$ (i.e. the flow value is also the total amount of flow into the sink t.)

$$
\begin{aligned}
|f| &= f(s,V) \quad \text{(because } f(s,V) + f((V-\{s\}),V) = f(V,V)\text{)} \\
&= f(V,V) - f((V-\{s\}),V) \quad \text{(because } f(V,V) = 0\text{)} \\
&= 0 - f((V-\{s\}),V) = -f((V-\{s\}),V) \text{(because } f(X,Y) = -f(Y,X)\text{)} \\
&= f(V,(V-\{s\})) \\
&= f(V,t) + f(V,(V-\{s,t\})) \\
&= f(V,t) + \textstyle\sum_{u\in V, u\neq s,t} f(V,u) \quad \text{(because } f(V,u) = 0 \text{ for all } u \neq s,t\text{)} \\
&= f(V,t) + \textstyle\sum_{u\in V, u\neq s,t} 0 \\
&= f(V,t)
\end{aligned}
$$

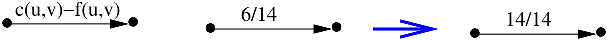
How to find a max-flow function?

Ford-Fulkerson Algorithm (outline)

1. Initialize a 0 flow function f, where $f(e) = 0$ for all $e \in E$.
2. **while** there's a path P from s to t that can be used to increase the flow value **do:**
3. augment flow f along P.
4. **end while**
5. **return** f

How an edge $e = u \to v$ can be used to increase the flow?

Case 1: $f(u \to v) \geq 0$. Then $c(u \to v) - f(u \to v)$ additional flow can go through e from u to v.

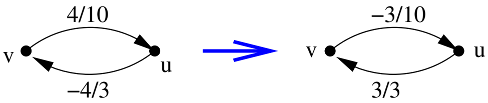


In this example, the capacity is $c(e) = 14$, the current flow is $f(e) = 6$. It is possible we can increase the flow by $c(e) - f(e) = 14 - 6 = 8$. (This is the maximum amount of additional flow that can be pushed through e without exceeding the capacity.)

Max-Flow: Algorithm

Case 2: $f(u \to v) < 0$. Then $a = f(v \to u) = -f(u \to v) > 0$. We can do two things:

- Cancel the a units flow from v to u.
- Push $c(u \to v)$ units flow from u to v.
- The net effect is that we can push $c(u \to v) + a$ units flow from u to v. which is $c(u \to v) - f(u \to v)$.



In this example $f(u \to v)$ is changed from -4 to 3. The net change is $7 = 3 - (-4) = c(u \to v) - f(u \to v)$.

Max-Flow: Algorithm

Residual Network

Let $G = (V, E)$ be a flow network and $f(*)$ a flow function of G. The residual network of G (with respect to f) is $G_f = (V, E_f)$ where:

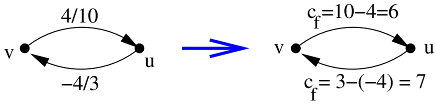
$$E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}$$

where

$$c_f(u, v) = c(u, v) - f(u, v)$$

is called the residual capacity of (u, v).

Note: $c_f(u, v)$ is the maximum amount of additional flow that can be pushed from u to v.



Max-Flow: Algorithm

Augmenting Path

- Let $G = (V, E)$ be a flow-network, f a flow function of G.
- Let G_f be the residual network of G with respect to f.
- Let P be a path in G_f from s to t. P is called an augmenting path of G_f.
- Define: $c_f(P) = \min\{c_f(u, v) \mid (u, v) \text{ is an edge on } P\}$.
- Define a flow function $f_P(*)$ on G_f by:

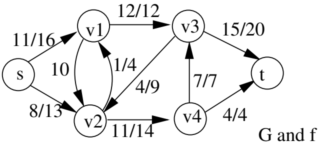
$$f_P(u,v) = \begin{cases} c_f(P) & \text{if } (u,v) \text{ is on } P \\ -c_f(P) & \text{if } (v,u) \text{ is on } P \\ 0 & \text{otherwise} \end{cases}$$

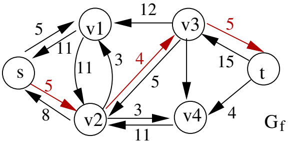
- Define a new flow function $f'(*)$ of G by:

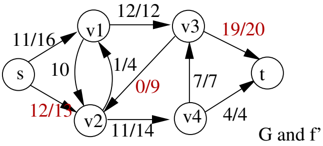
$$f'(e) = f(e) + f_P(e) \qquad \text{for all } e \in E$$

Max-Flow: Algorithm

- Along the path P, we can push more flow from s to t.
- $c_f(P)$ is the maximum amount of additional flow we can push along P. This is because there is an edge (u, v) on P whose residual capacity is $c_f(P)$. This is the bottle neck edge.
- $f'(*)$ is a new flow function with $|f'| = |f| + |f_P| = |f| + c_f(P)$. In other words, we increased the flow value by the amount $c_f(P)$.







augmenting path P

|f| = 11+8 = 19

c_f(P) = min {5,4,5} = 4

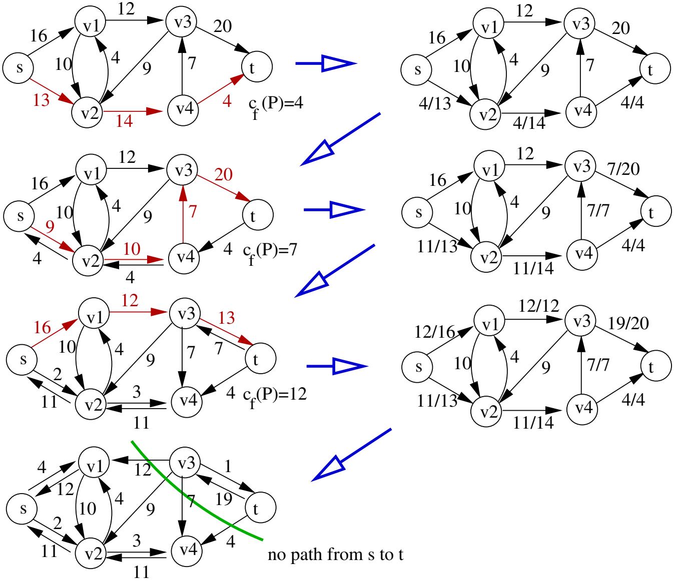
|f'| = 11+12 = 23

Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

for each $e = (u,v) \in E$ **do**
    $f(u,v) = f(v,u) = 0$
while there's a path P from s to t in G_f **do:**
    $c_f(P) = \min\{c_f(u,v) \mid (u,v) \text{ is on } P\}$
    for each $e = (u,v)$ on P **do**
        $f(u,v) = f(u,v) + c_f(P)$
        $f(v,u) = -f(u,v)$
end while
return f

Max-Flow: Example



Proof of Correctness

We will show:

Theorem

When Ford-Fulkerson algorithm terminates, f is a max-flow function of G.

We need a few definitions first.

Definition

Let $G = (V, E)$ be a flow network with source s and sink t. A cut of G is a partition of the vertex set V into two subsets S and T such that:

- $S \cap T = \emptyset$ and $S \cup T = V$.
- $s \in S$ and $t \in T$.
- The capacity of the cut is: $c(S,T) = \sum_{u \in S, v \in T, (u \to v) \in E} c(u,v)$
- The flow cross the cut is: $f(S,T) = \sum_{u \in S, v \in T, (u \to v) \in E} f(u,v)$

Proof of Correctness

Lemma (26.5)

Let G be a flow network with source s and sink t. Let f be a flow function of G. Let (S, T) be any cut of G. Then:

$$f(S,T) = |f|$$

Proof.

$$\begin{array}{lcll}
f(S,T) & = & f(S,V) - f(S,S) & \text{(because } S \cup T = V \; S \cap T = \emptyset\text{)} \\
 & = & f(S,V) & \text{(because } f(S,S) = 0\text{)} \\
 & = & f(s,V) + f((S - \{s\}), V) & \text{(because } S = \{s\} \cup (S - \{s\})\text{)} \\
 & = & f(s,V) & \text{(because the flow conservation} \\
 & & & \text{constraint, the 2nd term is 0)} \\
 & = & |f| &
\end{array}$$

□

Proof of Correctness

Lemma

Let G be a flow network with source s and sink t. Let f be a flow function of G. Let (S, T) be any cut of G. Then:

$$|f| \leq c(S, T)$$

Proof.

$$\begin{array}{rcll} |f| & = & f(S, T) & \text{(because Lemma 26.5)} \\ & = & \sum_{u \in S, v \in T, (u \to v) \in E} f(u, v) & \text{(by definition of } f(S, T)\text{)} \\ & \leq & \sum_{u \in S, v \in T, (u \to v) \in E} c(u, v) & \text{(by capacity constraint)} \\ & = & c(S, T) & \text{(by definition of } c(S, T)\text{)} \end{array}$$

□

Proof of Correctness

Max-Flow Min-Cut Theorem (26.6)

The following three statements are equivalent:

1. f is a max flow of G.
2. The residual network G_f contains no $s \to t$ path.
3. $|f| = c(S, T)$ for some cut (S, T) of G.

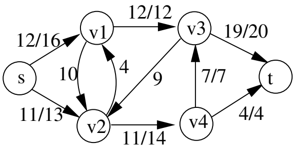
Proof (1) $\Rightarrow$ (2): Suppose (2) is not true. Then we can find an augmenting path P in the residual network G_f. We can push more flow along P to increase the flow value of f. Then f is not a maximum flow.
(2) $\Rightarrow$ (3): Define: $S = \{v \in V \mid$ there is a path in G_f from s to $v\}$ and $T = V - S$. We show (S, T) is a cut of G:

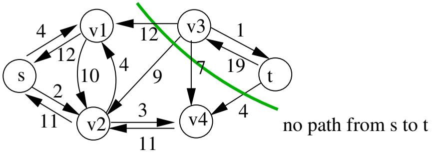
- $s \in S$: This is trivial because s itself is a path from s to s.
- $t \in T$: Because the condition (2), $t \notin S$. Hence $t \in T$.
- By the definition of T, we have $S \cap T = \emptyset$ and $S \cup T = V$.

Hence (S, T) is a cut of G.

Proof of Correctness

Proof (continued): For each edge $u \to v$ with $u \in S$ and $v \in T$, we have $f(u,v) = c(u,v)$. This is because if $f(u,v) < c(u,v)$, then $u \to v$ would be an edge in G_f with residual capacity $c(u,v) - f(u,v) > 0$. That means there is a path $s \to u \to v$ in G_f. This contradicts to the fact that $v \in T$.
Therefore $|f| = f(S,T) = c(S,T)$.





Proof (continued) (3) $\Rightarrow$ (1):

- By Lemma 26.6, for any flow function f and any cut (S,T) of G, we must have $|f| \leq c(S,T)$
- Now we have a flow f and a cut (S,T) such that $|f| = c(S,T)$.
- Then f must be a max-flow, and (S,T) must be a min-capacity cut.

Notes

- The equivalence of (1) and (3) says: For any flow network, the value of a max-flow is equal to the capacity of a min-cut. Hence, this theorem is called Max-Flow Min-Cut Theorem.
- This theorem is a fundamental theorem in mathematics. It appears in different branches, in different forms, by different names.

Proof of Correctness

Now we can prove:

Theorem

When Ford-Fulkerson algorithm terminates, f is a max-flow function of G.

Proof.

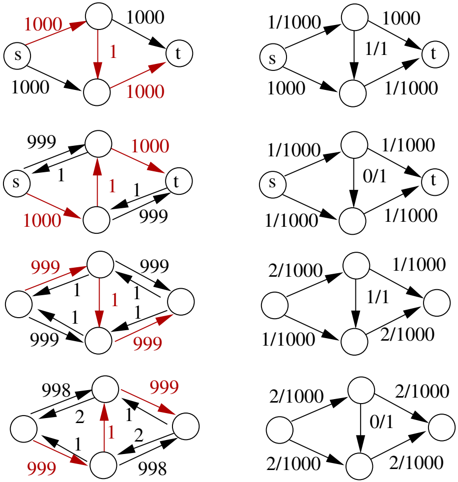
The algorithm stops only when there is no $s \rightarrow t$ path in G_f can be found. By Max-Flow Min-Cut Theorem (the equivalence of the statements (1) and (2)), f is a max flow. $\square$

Are we done yet?

No! There's a catch here: When the algorithm terminates ...
How do we know the algorithm will stop?

Proof of Correctness

- If all capacities are integers, then each iteration of the algorithm will increase the flow by at least 1.
- Since the flow value cannot exceed $c(s, V)$, the algorithm will stop.
- However, if we don't pick the augmenting path P carefully, the algorithm may take many iterations. In the following example:
  - The max flow value is clearly 1000+1000=2000,
  - If we pick augmenting path as shown, each iteration increases flow by 1.
  - So it takes 2000 iterations.
- If the capacities can be real numbers, there are examples of flow network and sequence of bad choices of augmenting paths for which the algorithm will never stop!



1000
1000
s
1
t
1000
1000
1/1000
1000
s
1/1
t
1000
1/1000
999
1000
s
1
1
t
1
1000
999
1/1000
1/1000
s
0/1
t
1/1000
1/1000
999
999
1
1
1
1
1
999
999
2/1000
1/1000
1/1
1/1000
2/1000
998
999
2
1
1
2
1
999
998
2/1000
2/1000
0/1
2/1000
2/1000

Karp-Edmonds Algorithm

It is identical to the Ford-Fulkerson's algorithm, except that

Karp-Edmonds Algorithm

When trying to find an augmenting path P in G_f, we must find a shortest path from s to t in G_f.

Here the shortest path means the path with smallest number of edges. We can use BFS on G_f for this.

Karp-Edmonds Theorem

The while loop in Karp-Edmonds Algorithm runs at most $|V| \cdot |E| = nm$ iterations.

Karp-Edmonds Algorithm: Analysis

Theorem

Karp-Edmonds Algorithm solves the max flow problem in $O(nm^2)$ time.

- In each iteration:
  - Construct G_f. This takes $O(n+m)$ time.
  - Find a shortest $s \to t$ path, by calling BFS. This takes $O(n+m)$ time.
  - All other operations take at most $O(n)$ time.
- The loop iterates $O(nm)$ time.
- So the total run time is $O(nm(n+m)) = O(nm^2)$. (This is because $n \leq m$).

- Note: Max-flow is a fundamental problem. Great efforts have been made to reduce the runtime. But for general cases, only small improvements have been made.

Application: Maximum Matching

Definition

Let $G = (V, E)$ be an undirected graph.

- A matching is a subset of edges $M \subseteq E$ such that no two edges in M share a common end vertex.
- A matching M is a maximal matching if, for any $e \notin M$, $M \cup \{e\}$ is not a matching of G.
- A matching M is a maximum matching if the number of edges in M is the largest among all matchings of G.
- A matching M is a perfect matching if every vertex u in G is incident to an edge in M.

Application: Maximum Matching

- In Fig 1 below, the blue edges form a maximal matching of G. (If we add any edge $e \notin M$ into M, it will no longer be a matching.) But it is not a maximum matching.
- In Fig 2, the set of blue edges is a maximum matching. It is also a perfect matching of G.

1) A maximal matching of G
But not a maximum matching

2) A maximum matching of G.

Note 1: A perfect matching is always a maximum matching. The reverse is not true.

Note 2: A maximum matching is always a maximal matching. The reverse is not true.

Maximum Matching (MM) Problem

Input: An undirected graph $G = (V, E)$.
Find: A maximum matching of G.

- This problem is much harder than it looks. It can be solved in polynomial time, but quite complicated.
- If the problem is to find a maximal matching, it can be solved by the following easy algorithm:

Maximal Matching$(G = (V, E))$

1. $M \leftarrow \emptyset$
2. **while** $E \neq \emptyset$ **do**
3. pick any edge e in E, add e into M
4. delete from E all edges that share a common vertex with e
5. **output** M

Maximum Matching

We are interested in solving the MM problem (i.e. finding a maximum matching) for a special case.

Definition

An undirected graph $G = (V, E)$ is called a bipartite graph if the vertex set V can be partitioned into two subsets X and Y such that:

- $X \cup Y = V$ and $X \cap Y = \emptyset$.
- Every edge of E connects a vertex in X and a vertex in Y.

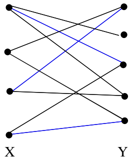
We use $G = (X, Y, E)$ to represent a bipartite graph.

Maximum Matching for Bipartite Graph (MMBG) Problem

Given a bipartite graph $G = (X, Y, E)$, find a maximum matching of G.

MMBG Problem

The following Fig shows a bipartite graph $G = (X, Y, E)$ and a matching M of G. Is M maximum? It's not clear.



Application 1 of MMBG: Marriage Problem

Given a bipartite graph $G = (X, Y, E)$:

- $X = \{x_1, x_2, \ldots, x_p\}$ is the set of boys.
- $Y = \{y_1, y_2, \ldots, y_q\}$ is the set of girls.
- $(x_i, y_j) \in E$ means x_i and y_j like each other.
- A MM M of G represents a maximum set of marriages.
- If you run an on-line match-making web-site, you need this algorithm.

MMBG Problem

Application 2 of MMBG: Distinct Representative Problem

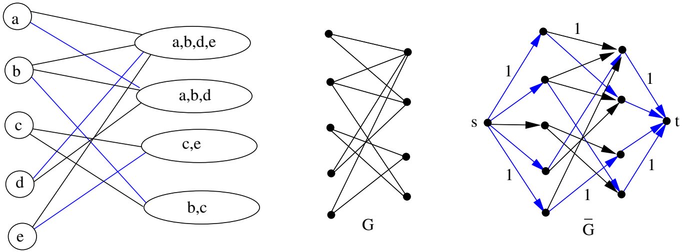
- UB has p student clubs $C_1, C_2, \ldots, C_p$.
- A student may be a member of several clubs.
- Need to select a committee so that:
  - Each club has one representative in the committee.
  - Each student in the committee represents only one club (even if he/she is a member of multi-clubs.)

How to do this?

This is a MMBG problem. Define a bipartite graph $G = (X, Y, E)$:

- $X = \{C_1, C_2, \ldots, C_p\}$ (each element of X is a club.)
- $Y = \{y_1, y_2, \ldots, y_q\}$ (each element of Y is a student.)
- $(C_i, y_j) \in E$ if and only if y_j is a member of C_i.
- Find a MM M of G. If every C_i is incident to an edge in M, then we can select the committee. If not, this is impossible.

MMBG Problem



Converting MMBG to Max-Flow

Given an input instance $G = (X, Y, E)$ of MMBG, we construct a flow network $\bar{G}$ as follows:

- $V(\bar{G}) = X \cup Y \cup \{s, t\}$
- $E(\bar{G}) = \{s \to x \mid \forall x \in X\} \cup \{y \to t \mid \forall y \in Y\} \cup \{x \to y \mid \forall (x, y) \in E\}$
- All edges have capacity 1.

MMBG Problem

Lemma

Let M be a MM of $G = (V, E)$. Let f be a max-flow function of $\bar{G}$. Then $|M| = |f|$.

Proof.

Let M be a MM of G. Suppose $M = \{e_1, e_2, \ldots, e_k\}$, where $e_i = (x_i, y_i)$. Let $f_M(*)$ be defined as follows:

- $f_M(s \to x_i) = 1$ for all $1 \leq i \leq k$
- $f_M(y_i \to t) = 1$ for all $1 \leq i \leq k$
- $f_M(x_i \to y_i) = 1$ for all $1 \leq i \leq k$
- $f_M(e) = 0$ for all other edges.

It is easy to check f_M is a flow function of $\bar{G}$ and $|f_M| = k$. Since f is a max flow, we have $|f| \geq |f_M| = k = |M|$. □

MMBG Problem

Proof.

Conversely, let f be a max flow function of $\bar{G}$. Because all edge capacities of $E(\bar{G})$ are 0/1, it is easy to see that $f(e)$ is 0/1 for all edges e. Define:

$$M_f = \{(x,y) \mid x \in X, y \in Y \text{ and } f(x \to y) = 1\}$$

We show M_f is a matching of G. It's enough to show each vertex of G is incident to at most one edge in M_f. Suppose $M_f = \{(x_1,y_1),(x_2,y_2),\ldots,(x_t,y_t)\}$. Consider an edge $(x_i,y_i) \in M_f$.

- $(x_i,y_i) \in M_f$ is because $f(x_i \to y_i) = 1$.
- Because $f(e) = 0$ or 1 for all edges, and the flow conservation constraint at x_i, we must have: (a) $f(s \to x_i) = 1$ and (b) $f(x_i,y_j) = 0$ for all $y_j \neq y_i$.
- So $(x_i,y_j) \notin M$ for all $j \neq i$. Namely x_i is incident to exactly one edge in M_f
- Similarly, we can show y_i is incident to exactly one edge in M_f.

So M_f is a matching of G. But M is a MM of G. Thus $|M| \geq |M_f| = t = |f|$.

□

MMBG Problem: Time Analysis

- The conversion from G to $\bar{G}$ can be done in $O(n+m)$ time.
- Run Karp-Edmonds algorithm on $\bar{G}$ to find a max-flow f of $\bar{G}$ takes $\Theta(nm^2)$ time.
- Constructing an MM of G from f takes $O(n)$ time.
- So the entire process takes $O(nm^2)$ time.
- However, $\bar{G}$ is a very special flow network: All edge capacities are 1; the length of the longest $s \to t$ path is only 3.
- For this kind network, the max-flow algorithm runs much faster.
- Currently, the best algorithm for solving MMBG is Karp-Hopcroft algorithm, with runtime $O(n^{1/2}m)$.

Weighted MM and MMBG Problem

Weighted MM Problem

Input: An undirected graph $G = (V,E)$. Each edge $e \in E$ has a weight $w(e) \geq 0$.
Find: A matching M of G so that the total weight of M $w(M) = \sum_{e \in M} w(e)$ is maximum.

The MM problem is just a special case of the Weighted MM problem:

$$w(e) = 1 \text{ for all } e \in E$$

Weighted MMBG Problem

The weighted version of the MMBG problem

Weighted MMBG Problem: Application

The Weighted MMBG Problem is also called:

Personnel Assignment Problem

A bipartite edge weighted graph $G = (X, Y, E)$ represents the following:

- $X = \{x_1, x_2, \ldots, x_p\}$ is the set of workers.
- $Y = \{y_1, y_2, \ldots, y_p\}$ is the set of jobs.
- $(x_i, y_j) \in E$ means the worker x_i can do the job y_j.
- $w(x_i, y_j)$ is the profit we get if x_i is assigned to job y_j.
- We assume each job only needs one worker and one worker can only be assigned to one job.

Problem: How to assign the workers to jobs in order to maximize total profit.

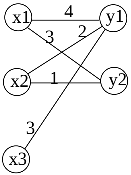
A maximum weight matching M of G is the optimal worker assignment.

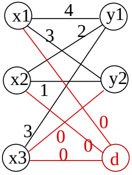
Weighted MMBG Problem: Application

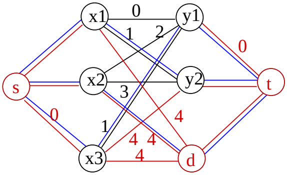
This problem can be converted to the min-cost max-flow problem. We are given an instance of weighted MMBG problem: $G = (X, Y, E)$ with weight function $w(*)$. We construct a flow network with edge cost as follows.

- First, we add dummy vertices into either X or Y and edges into G so that it is a complete bipartite graph G_1 with $|X| = |Y|$.
  - Complete bipartite graph means: for any $x \in X$ and $y \in Y$, $(x, y) \in E$.
  - If $(x, y) \notin E$, add a dummy edge (x, y) into E with $w(x, y) = 0$.
  - Interpretation: $(x, y) \notin E$ means the worker x cannot do the job y. Adding (x, y) means we pretend x can do y. But $w(x, y) = 0$ means we get no profit if x is assigned to y. So nothing is really changed!
- Then, we construct a flow-network $\bar{G}_2$ from G_1.
  - Add a new source s and a new sink t.
  - For each $x \in X$, add a new edge (s, x). For each $y \in Y$, add a new edge (y, t). All these edges have capacity $c(s, x) = c(y, t) = 1$ and $cost(s, x) = cost(y, t) = 0$.
  - Let K be the largest $w(e)$ value for all edges e in G_1. For each edge e in G_1, define capacity $c(e) = 1$ and $cost(e) = K - w(e)$.

Weighted MMBG Problem: Application







- $G = (X, Y, E)$ is the input weighted bipartite graph. $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2\}$.
- G_1 is a complete bipartite graph. d is a dummy vertex. The red edges are dummy edges. They all have $w(e) = 0$. The max weight is $K = 4$.
- $\bar{G}_2$ is the flow-network constructed from G_1. All edges have capacity 1. The edges adjacent to s and t have $cost = 0$.. The cost of other edges are as marked. The flow on each of the three blue paths is 1. The flow on all other edges are 0. The corresponding assignment is: x_1 is assigned to y_2. x_3 is assigned to y_1. x_2 has no real assignment.

Weighted MMBG Problem: Application

We can argue this procedure indeed solves the personnel assignment problem. Let f be the min-cost max-flow function of G_2.

- Suppose $|X| = |Y| = t$ in G_1 (and in $\bar{G}_2$).
- Since all edges in $\bar{G}_2$ have capacity $c(e) = 1$, and G_1 is a complete bipartite graph, the flow f consists of t edge disjoint paths from s to t in $\bar{G}_2$, and each path carries 1 unit flow.
- These paths restricted to the edges in G_1 form a perfect matching M_1 of G_1 with minimum total cost. Because all edges $s \to x$ and $y \to t$ have 0 cost, they do not contribute to the total cost.
- For each edge e in G_1, $cost(e) = K - w(e)$. So minimizing cost is the same as maximizing the profit (i.e. weight $w(e)$.)

Connectivity Problems

Vertex Connectivity

Let $G = (V, E)$ be an undirected graph. The vertex connectivity of G, denoted by $\kappa(G)$, is the minimum number of vertices that must be deleted from G in order to disconnect G.

- $\kappa(G) = 0$ if and only if G is not connected.
- $\kappa(G) = 1$ if and only if G is connected and has at least one cut vertex.
- $\kappa(G) \geq 2$ if and only if G is biconnected.

Application

- G represents a computer network. Each vertex is a computer site. Each edge $e = (x, y)$ is a communication line between site x and site y.
- The network can survive a k-site failure if and only if the vertex connectivity of G is at least $k + 1$.

Connectivity Problems

Edge Connectivity

Let $G = (V, E)$ be an undirected graph. The edge connectivity of G, denoted by $\kappa'(G)$, is the minimum number of edges that must be deleted from G in order to disconnect G.

Application

- G represents a computer network. Each vertex is a computer site. Each edge $e = (x, y)$ is a communication line between site x and site y.
- The network can survive a k-link failure if and only if the edge connectivity of G is at least $k + 1$.

The vertex connectivity and the edge connectivity can also be defined for directed graphs with similar meaning.

Connectivity Problems

$\kappa(G)$ and $\kappa'(G)$ are two fundamental parameters for G. How to find them?

- For undirected graph, we can find if $\kappa(G) \geq 0$ by using BFS in $O(n+m)$ time.
- We can find if $\kappa(G) \geq 1$ (namely, if G has a cut vertex or not) by using DFS in $O(n+m)$ time.
- We can find if $\kappa(G) \geq 2$ by a much more complicated application of DFS in $O((n+m)\log n)$ time. (We do dot discuss details here).
- But for a general k, how do we determine if $\kappa(G) \geq k$ or not?

Connectivity Problems

Brute-Force-Vertex-Connectivity$(G = (V, E))$

1. Enumerate all subsets $C \subset V$.
2. **for** each $C \subset V$ generated **do**
3. delete C from G
4. test if $G - C$ is connected
5. output the minimum size C where $G - C$ is disconnected

However, there are 2^n vertex subsets. This would take $\Omega(2^n)$ time.

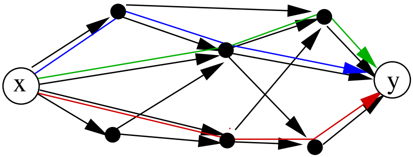
Connectivity Problems: Using Max-Flow

We consider the computation of $\kappa'(G)$ for directed graph G first.

Definition

Let $G = (V, E)$ be a directed graph and x, y two vertices of G.

$$\begin{aligned} \kappa'_G(x,y) &= \text{the minimum number of edges that must be deleted} \\ &\quad\ \text{from } G \text{ in order to disconnect } x \text{ from } y. \\ &= \text{the maximum number of edge disjoint paths from } x \text{ to } y. \end{aligned}$$



Connectivity Problems

Note: The equivalence of the two definitions of $\kappa'_G(x, y)$ is another form of the max-flow min-cut theorem.

$\kappa'_G(x, y)$ can be computed as follows:

- Treat $G = (V, E)$ as a flow network.
- x is the source and y is the sink.
- All edges have capacity $c(e) = 1$.
- Find a max flow f for G. Then $|f| = \kappa'_G(x, y)$.

Connectivity Problems

Theorem

$$\kappa'(G) = \min_{x,y \in V, x \neq y} \kappa'_G(x,y)$$

Edge-Connectivity Algorithm

1. **for** each pair of vertices $x, y \in V$ **do**
2.   compute $\kappa'_G(x,y)$
3. **output** the smallest $\kappa'_G(x,y)$

Fact

Let $T(n,m)$ be the run time for solving the max-flow problem for this special case. (Because of the special structure of the input, it is easier than the general max-flow problem). Then the edge connectivity problem can be solved in $\Theta(n^2 T(n,m))$ time.

Connectivity Problems

The edge connectivity problem for undirected graph is very similar.

Definition

Let $G = (V, E)$ be an undirected graph and x, y two vertices of G.

$\kappa'_G(x, y)$ = the minimum number of edges that must be deleted from G in order to disconnect x and y.
= the maximum number of edge disjoint paths between x and y.

We can calculate $\kappa'_G(x, y)$ by using max-flow algorithm. The only difference is that $G = (V, E)$ is an undirected network with source x and sink y. Then we can change it to the basic max-flow problem as we discussed before.

Theorem

$$\kappa'(G) = \min_{x,y \in V, x \neq y} \kappa'_G(x, y)$$

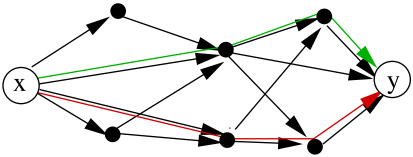
Then the edge connectivity problem for undirected graph can be solved in $\Theta(n^2 T(n, m))$ time.

We consider the vertex connectivity problem for directed graphs.

Definition

Let $G = (V, E)$ be a directed graph and x, y two vertices of G.

$\kappa_G(x,y)$ = the minimum number of vertices that must be deleted from G in order to disconnect x and y.
= the maximum number of vertex disjoint paths between x and y.



Connectivity Problems

Note: The equivalence of the two definitions of $\kappa_G(x,y)$ is yet another form of the max-flow min-cut theorem.

$\kappa_G(x,y)$ can be computed as follows:

- Treat $G=(V,E)$ as a directed flow network.
- x is the source and y is the sink.
- All edges have capacity $c(e)=1$.
- All vertices $u \neq x,y$ have vertex capacity $c(u)=1$.
- Find a max flow f for G. Then $|f| = \kappa'_G(x,y)$.

- So this is the max-flow problem for directed network, with both edge and vertex capacities.
- It can be converted to the basic max-flow problem as discussed before.
- $\kappa_G(x,y)$ can be computed in $\Theta(T(n,m))$ time.
- $\kappa(G) = \min_{x,y \in V} \kappa_G(x,y)$ can be computed in $\Theta(n^2 T(n.m))$ time.
- The vertex connectivity problem for directed graph is almost identical