Lecture 4: Flow Algorithms

Piotr Sankowski sankowski@dis.uniroma1.it
Stefano Leonardi leon@dis.uniroma1.it

Theoretical Computer Science ’ 22.10.2009

Lecture Overview

- Flows: Problem Definition
- Ford-Fulkerson Algorithm
- Edmonds-Karp Algorithm
- Blocking Flow Algorithms
  - Dinic Algorithm
  - "Three Indians" Algorithm

Flows: Problem Definition

A flow network or shortly *network* is a directed graph $G = (V, E)$ together with the function $c : E \to \mathcal{R}_+$ that gives capacities of the edges.

In a flow network we distinguish two vertices:

- *source* denoted as s,
- *sink* denoted as t.

A flow in G is a function $f : V \times V \to \mathcal{R}$.

Flows: Problem Definition

A flow in G is a function $f : V \times V \to \mathcal{R}$ fulfilling the following conditions:

- for every $u, v \in V$ we have $f(u,v) \leq c(u,v)$ – capacity condition,
- for every $u, v \in V$ we have $f(u,v) = -f(v,u)$ – *skew symmetry* condition,
- for every $u \in V - \{s,t\}$ we require

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

flow conservation condition.

Flows: Problem Definition

We say that the value $f(u,v)$ is the flow from u do v.

The *value* of the flow f is denoted $|f|$ and defined as the total flow leaving s over all edges, i.e.:

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

In the *maximum flow problem* for a given network we want to find the flow with the highest value.

Flows: Problem Definition

We use the following summing notation, where the argument for the function $f : X \rightarrow Y$ can be a subset A of X.

In such a case we assume that the value of f is the sum of its values for set A, i.e.:

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

Using this notation we can write the flow conservation condition for the vertex v as $f(v,V) = 0$.

Flows: Problem Definition

Lemma 1 *Let $G = (V, E)$ be the flow network and let f be a flow in it then:*

- *for every $X \subseteq V$ we have $f(X, X) = 0$,*
- *for every $X, Y \subseteq V$ we have $f(X, Y) = -f(Y, X)$,*
- *for every $X, Y, Z \subseteq V$ we have:*

$$f(X \cup Y, Z) = f(X, Z) + f(Y, Z) - f(X \cap Y, Z)$$

$$f(Z, X \cup Y) = f(Z, X) + f(Z, Y) - f(Z, X \cap Y).$$

- $|f| = f(s, V) = f(V, t)$.

We prove this lemma during exercises.

Residual Network

Intuitively, for given G and the flow f the residual network contains edges that can take more flow.

More formally, for a given pair of vertices $u, v \in V$ the additional flow $c_f(u,v)$ that can be send from u to v till we reach the capacity $c(u,v)$ is

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

For example, if $c(u,v) = 16$ and $f(u,v) = 11$ then we can increase $f(u,v)$ by $c_f(u,v) = 5$ till we violate the capacity of the edge (u,v).

Residual Network

For the given network $G = (V, E)$ and flow f, the residual network G_f induced by f is the graph $G_f = (V, E_f)$ such that:

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

In other words for each edge in the residual network we can still increase the flow by more then 0.

The edges in E_f can be directed as edges in E or have reversed direction.

Residual Network

If $f(u,v) < c(u,v)$ for $(u,v) \in E$ then
$c_f(u,v) = c(u,v) - f(u,v) > 0$ and $(u,v) \in E_f$.

Whereas if $f(u,v) > 0$ for $(u,v) \in E$ then $f(v,u) < 0$. In such case $c_f(v,u) = c(v,u) - f(v,u) > 0$, so $(v,u) \in E_f$.

Of course if (u,v) nor (v,u) are in G then
$c(u,v) = c(v,u) = 0$ and $f(u,v) = f(v,u) = 0$, so
$c_f(u,v) = c_f(v,u) = 0$ as well.

Hence, $|E_f| \leq 2|E|$. Note that the residual network G_f is a flow network with capacities given by c_f.

Residual Network

The following lemma shows the connection between the residual network and the original network.

Lemma 2 *Let $G = (V, E)$ be the flow network with source s and sing t, and let f be a flow in G. Moreover, let f' be a flow in the residual network G_f. Then flow sum $f + f'$ is a valid flow in G with value $|f + f'| = |f| + |f'|$.*

We need to show that the flow conditions hold for the sum, i.e.: skew symmetry condition, capacity condition and the flow conservation condition.

Residual Network

For the skew symmetry condition note that, for all $u, v \in V$, we have:

$$\begin{aligned}(f + f')(u,v) &= f(u,v) + f'(u,v) = \\ &= -f(v,u) - f'(v,u) = \\ &= -(f(v,u) + f'(v,u)) = -(f + f')(v,u).\end{aligned}$$

For the capacity condition note that $f'(u,v) \leq c_f(u,v)$, for all $u, v \in V$, and so:

$$\begin{aligned}(f + f')(u,v) &= f(u,v) + f'(u,v) \leq \\ &\leq f(u,v) + (c(u,v) - f(u,v)) = c(u,v).\end{aligned}$$

Residual Network

For the flow conservation condition, for all $u \in V - s, t$, we have:

$$\begin{aligned}\sum_{v \in V}(f + f')(u, v) &= \sum_{v \in V}(f(u, v) + f'(u, v)) = \\ &= \sum_{v \in V} f(u, v) + \sum_{v \in V} f(u, v) = 0 + 0 = 0.\end{aligned}$$

Finally, the value of $f + f'$ is:

$$\begin{aligned}|f + f'| &= \sum_{v \in V}(f + f')(s, v) = \\ &= \sum_{v \in V}(f(s, v) + f'(s, v)) = \\ &= \sum_{v \in V} f(s, v) + \sum_{v \in V} f'(s, v) = |f| + |f'|.\end{aligned}$$

Augmenting Paths

For the flow network $G = (V, E)$ and the flow f, the *augmenting path* p is a simple path from s to t in the residual network G_f.

By the definition of the residual network for every edge (u, v) on the augmenting we can increase the flow from u to v without violating the capacity condition.

The highest increase of the value of the flow f that can achieved using the path p is the *residual capacity* of p:

$$c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is on } p\}.$$

Augmenting Paths

Finding augmenting paths.

Zasd_ilustr_l.swf

Augmenting Paths

The increase of the flow for p is a function $f_p : V \times V \rightarrow R$:

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p, \\ -c_f(p) & \text{if } (v,u) \text{ is on } p, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Note that f_p is a flow in G_f with value $|f_p| = c_f(p) > 0$.

Corollary 1 *Let $f' = f + f_p$ then f' is a flow in G with value $|f'| = |f| + |f_p| > |f|$.*

Minimum Cuts

The maximum flow and minimum cut theorem, we will show, says that the flow is maximum if and only if the residual network does not contain augmenting path.

"'Cut"' (S, T) in the flow network $G = (V, E)$ is division of V into S and $T = V - S$ such that $s \in S$ and $t \in T$.

If f is a flow then *netto flow* through the cut (S, T) is defined as $f(S, T)$.

Capacity of the cut (S, T) is defined as $c(S, T)$.

Minimum cut is the one with minimum capacity.

Minimum Cuts

The cut $(\{s, v_1, v_2\}, \{v_3, v_4, t\})$ with netto flow $f(v_1, v_3) + f(v_2, v_3) + f(v_2, v_4) = 12 + (-4) + 11 = 19$, and capacity $c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26$.

Zasd_ilustr_b.swf

Minimum Cuts

One should note that netto flow through the cut can contain negative flow, but the capacity of the cut contains only nonnegative values.

In other words netto flow through the cut (S, T) is obtained by:

- adding the positive flow from S to T,
- subtracting the positive flow from T to S.

On the other hand the capacity of the cut (S, T) is composed only out of the edges going from S to T. The edges going from T to S are not taken into account.

Minimum Cuts

The following lemma shows that the netto flow through any cut is exactly equal to the value of the flow.

Lemma 3 *Let f be a flow in the network G and let (S,T) be the cut in G then netto flow through (S,T) is equal to $f(S,T) = |f|$.*

Note that $f(S-s,V) = 0$ by flow conservation condition, so:

$$\begin{aligned} f(S,T) &= f(S,V) - f(S,S) = \\ &= f(s,V) + f(S-s,V) = f(s,V) = |f|. \end{aligned}$$

Minimum Cuts

A a result we get that the capacity of the cut is an upper bound on the flow value.

Corollary 2 *The value of any flow f in G is bounded from above by the capacity of any cut in G.*

From Lemma 3 and the capacity condition we get:

$$|f| = f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) \leq$$
$$\leq \sum_{u \in S} \sum_{v \in T} c(u,v) = c(S,T).$$

Minimum Cuts

The maximum flow is upper bounded by the minimum cut.

The maximum flow and minimum cut theorem says that these values are actually equal.

Theorem 1 *For a flow f in a network $G = (V, E)$ the following conditions are equivalent:*

1. *f is the maximum flow in G,*
2. *residual network G_f does not contain augmenting paths,*
3. *$|f| = c(S, T)$ for some cut (S, T) in G.*

Minimum Cuts

(1)$\rightarrow$(2):
Assume by contradiction that f is the maximum flow but G_f contains an augmenting path p.

Consider the flow f' given as the sum $f' = f + f_p$ where f_p is given by (1).

By Corollary 1 f' is a flow in G and is strictly larger then f.

This contradicts the assumption that f is maximum.

Minimum Cuts

(2)$\rightarrow$(3):
Let us assume that G_f does not contain any augmenting path, i.e., there is no path going from s to t in G_f.

Define:

$$S := \{v \in V : \text{ there exists a path from } s \text{ to } v \text{ in } G_f\}$$

$$T := V - S$$

There is not path form s to t in so, $t \in T$ and trivially $s \in S$. Hence (S, T) is a cut.

Minimum Cuts

For every pair $u \in S$ and $v \in T$, we have $f(u,v) = c(u,v)$, as otherwise $(u,v) \in E_f$, and v would be in S.

By Lemma 3 we get $|f| = f(S,T) = c(S,T)$.

(3)$\rightarrow$(1):
By Corollary 2 we know that $|f| \leq c(S,T)$ for all cuts (S,T).

Hence the condition $|f| = c(S,T)$ means that f is the maximum flow.

Ford-Fulkerson Algorithm

In each iteration of FF we find an augmenting path.

- `for` every edge $(u,v) \in E$ `do`
  - $f(u,v) = 0$
  - $f(v,u) = 0$
- `while` there exists a path p from s to t in the residual network G_f `do`
  - $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
  - `for` every edge $(u,v) \in p$ `do`
    - $f(u,v) = f(u,v) + c_f(p)$
    - $f(v,u) = -f(u,v)$
- `return` f

Ford-Fulkerson Algorithm

The run of the algorithm is presented on the following figure.

Zasd_ilustr_r.swf

Ford-Fulkerson Algorithm

The running time of the Ford-Fulkerson algorithm depends on the choice of the augmenting path.

If we do it wrongly the algorithm might even not stop.

In the case when the edge capacities are integral, we can easily show that the running time of the algorithm is $O(E|f^*|)$, where f^* is the maximum flow.

If f^* is small the algorithm finishes fast, but even in easy cases it might need $|f^*|$ iterations.

Ford-Fulkerson Algorithm

Such an example is presented on the following figure.

zasd_ilustr_e.swf

Edmonds-Karp Algorithm

Edmonds-Karp algorithm is the Ford-Fulkerson algorithm in which instead of any path we use the shortest augmenting path.

We assume that the edges have unit lengths.

This modification allows to improve the running time of the algorithm to polynomial time.

We will show that Edminds-Karp algorithm works in $O(nm^2)$ time.

Edmonds-Karp Algorithm

We denote by $d_f(u,v)$ the distance from u to v in G_f – assuming that each edge has unit length.

Lemma 4 *If Edmonds-Karp algorithm works in flow network $G = (V,E)$ with the source s and sink t then for all vertices $v \in V - \{s,t\}$ the distance $d_f(s,v)$ in the residual network G_f does not decrease.*

Assume that for some vertex $v \in V - \{s,t\}$ there exists augmenting path q that decreases the distance from s to v.

Edmonds-Karp Algorithm

Let f be the flow before we apply the path p and let f' be the flow just after this.

Assume that v was the vertex minimizing $d_{f'}(s,v)$.

We have $d_{f'}(s,v) < d_f(s,v)$.

Let $p = s \to u \to v$ be the shortest path from s to v in $G_{f'}$ such that $(u,v) \in E_{f'}$ and

$$d_{f'}(s,u) = d_{f'}(s,v) - 1.$$

Edmonds-Karp Algorithm

Due to the choice of v we know that the distance to u was not decreased:

$$d_{f'}(s,u) \geq d_f(s,u).$$

We will prove that $(u,v) \notin E_f$. Assume otherwise $(u,v) \in E_f$, then using triangle inequality for s, v and u we get:

$$d_f(s,v) \leq d_f(s,u) + 1 \leq d_{f'}(s,u) + 1 \leq d_{f'}(s,v),$$

This contradicts the assumption that $d_{f'}(s,v) < d_f(s,v)$.

Edmonds-Karp Algorithm

We have $(u,v) \notin E_f$ and $(u,v) \in E_{f'}$.

Hence, the augmentation of the flow f to f' increased the flow v to u.

Edmonds-Karp uses shortest paths, so the shortest path from s to u in G_f ends with the edge (v,u) and:

$$\begin{aligned} d_f(s,v) &= d_f(s,u) - 1 \leq \\ &= d_{f'}(s,u) - 1 = d_{f'}(s,v) - 2, \end{aligned}$$

this contradict the assumption that $d_{f'}(s,v) < d_f(s,v)$.

Edmonds-Karp Algorithm

The following theorem bounds the number of iterations of Edmonds-Karp algorithm.

Theorem 2 *For a network $G = (V, E)$ Edmonds-Karp algorithm is executed at most $O(|V||E|)$ times.*

Every iteration of the algorithm can be implemented in $O(|E| + |V| \log |V|)$ time, so the total running time of the algorithm is $O(|E|^2|V| + |E||V|^2 \log |V|)$ time.

Edmonds-Karp Algorithm

We say that an edge (u, v) in residual network G_f is *critical* on an augmenting path p if residual capacity of p equals to the residual capacity of (u, v), i.e.:

$$c_f(p) = c_f(u, v).$$

After we increase the flow using the augmenting path every critical edge disappears from the residual network.

We will show that every edge from E can become critical at most $|V|/2 - 1$ times.

Edmonds-Karp Algorithm

Let u and v be connected by a critical edge edge.

The augmenting paths are the shortest paths so for the critical edge (u, v) we get

$$d_f(s, v) = d_f(s, u) + 1.$$

After the augmentation the edge (u, v) disappears and cannot appear on any other augmenting path till the flow from u to v is not decreased.

This can happen only if when (v, u) appears on an augmenting path.

Edmonds-Karp Algorithm

If f' is the flow in G and this happens then we have:

$$d_{f'}(s,u) = d_{f'}(s,v) + 1.$$

We have $d_f(s,v) \leq d_{f'}(s,v)$ by Lemma 4 we get:

$$\begin{aligned} d_{f'}(s,u) &= d_{f'}(s,v) + 1 \geq \\ &\geq d_f(s,v) + 1 = d_f(s,u) + 2. \end{aligned}$$

Hence, between the time when (u,v) was once critical and the becomes critical again the distance from the source to u hes to increase by at least 2.

Edmonds-Karp Algorithm

The distance to u at the beginning is bigger then 0.

The path from s to u cannot contain t, because (u, v) is critical and $u \neq t$.

The distance to u is at most $|V| - 2$.

Hence, (u, v) can become critical at most $(|V| - 2)/2 = |V|/2 - 1$ times.

Each augmenting path contains at least one critical edge, so the total number of such paths is $O(|V||E|)$.

Blocking Flows

A *blocking flow* in residual network G_f is a flow b in G_f, such that:

1. every path from s to t in b is a shortest path in G_f,
2. every shortest path in G_f contains a saturated edge in G_{f+b}.

Saturated edge is an edge with residual capacity equal to zero.

Note that this definition corresponds to the notion of the maximal set of shortest paths in Hopcroft-Karp algorithm.

Blocking Flow Algorithm

Dinic algorithm uses blocking flows to find the maximum flow in the following way.

- $f = 0$
- `while` there exists a path from s to t in G_f `do`
  - znajdź przepływ blokujący b w G_f
  - $f = f + b$
- `return` f

The correctness of the algorithm results directly from Maximum Flow and Minimum Cut theorem because when the algorithm stops there are no augmenting paths in G.

Blocking Flow Algorithm

Let us now see how many times the `while` loop can be executed.

Lemma 5 *Let b be the blocking flow in G_f then the shortest augmenting path in G_{f+b} is longer then the shortest augmenting path in G_f.*

Assume that the length of the shortest path p in G_{f+b} is not larger then the length of the shortest path in G_f.

Then p shares a saturated edges with the blocking flow b.

Blocking Flow Algorithm

Let $u - v$ be the last such edge on p.

This means that $v - u$ belongs to b. Otherwise in G_{f+b} the edge $u - v$ would still be saturated.

The flow b can be decomposed into the sum of shortest paths in G_f, so by Lemma 4 the distance from s to u did not decrease, i.e., $d_f(s,u) \leq d_{f+b}(s,u)$.

However, p is the shortest path from s to t so the distance to v could increase by at most 2, $d_f(s,v) + 2 \leq d_{f+b}(s,v)$.

Blocking Flow Algrithm

The part of p from v to t is the shortest path as well, so $d_f(s,t) + 2 \leq d_{f+b}(s,t)$, i.e., the length of the shortest path in residual graph has increased.

Corollary 3 *The length of the shortest path is at most $n-1$ so the maximum number of phases in Dinic algorithm is at most n.*

Finding a Blocking Flow

In order to find blocking we need the *layered network*.

Layered network $\overline{G}_f$ for the residual network $G_f = (V, E_f)$ is a directed graph $\overline{G}_f = (V, \overline{E}_f)$ with edges:

$$\overline{E}_f = \{(u,v) : (u,v) \in E_f \text{ and } d_f(s,u) + 1 = d_f(s,v)\}.$$

Note that all paths from s to t in $\overline{G}_f$ are shortest paths.

Hence, if we find a blocking flow in layered graph we will automatically fulfil the first condition.

Finding a Blocking Flow

In the graph $\overline{G}_f$ not all paths lead to t.

- $b = 0$
- construct the graph $\overline{G}_f$
- `while` $\overline{E}_f \neq \emptyset$ `do`
  - find a path p from s to t in $\overline{G}_f$
  - `for` each edge $(u,v) \in p$ `do`
    - $b(u,v) = b(u,v) + c_f(p)$
    - $b(v,u) = -b(u,v)$
    - recursively remove u and other vertices if no residual edge leaves them
- `return` b

Finding a Blocking Flow

The following animation shows the run of the algorithm

Zasd_ilustr_p.swf

Finding a Blocking Flow

Note that when the algorithm stops there is no path from s to t in $\overline{G}_f$, so the returned flow is a blocking flow.

The main loop in the algorithm will be executed at most $|E|$ times, as in each run at least one edge is saturated.

We can implement the loop to work in $O(n)$, so the total running time of the procedure requires $O(nm)$ time.

By Corollary 3 the total running time of Dinic algorithm is $O(mn^2)$.

”Three Indians” Algorithm

The ”three Indians” algorithm is called as well MKM algorithm (Malhotra, Kumar, Maheshwari).

In each execution of the main loop we will saturate one vertex by sending the flow backward and forward.

During the algorithm the function f will not fulfill the flow conditions. However, at the end of the algorithm these conditions will be restored.

Vertex capacity for $v \in V$ is defined as:

$$c(v) = \min\{\sum_{u \in V} c(u,v), \sum_{u \in V} c(v,u)\}.$$

"Three Indians" Algorithm

We need the following two auxiliary procedures:

- `FORWARD`(v) – if more flow enters v then leaves it, this procedure sends the surplus flow forward in $\overline{G}_f$ saturating one by one the edges leaving v,
- `BACKWARD`(v) – if more flow leaves v then enters it, this procedure compensates this insufficiency by sending flow on to v in $\overline{G}_f$ saturating one by one edges edges entering v.

"Three Indians" Algorithm

- $b = 0$ and construct the graph $\overline{G}_f$
- `while` $\overline{E}_f \neq \emptyset$ `do`
  - find a vertex with the smallest $c(v)$
  - send $c(v)$ units of flow using edges leaving v
  - send $c(v)$ units of flow using edges entering v
  - `for` $i = d(s,v) + 1$ `to` $n - 1$ `do`
    - `for` $w \in \{w \in V : d(s,w) = i\}$ `do` FORWARD(w)
  - `for` $i = d(s,v) - 1$ `downto` 1 `do`
    - `for` $w \in \{w \in V : d(s,w) = i\}$ `do` BACKWARD(w)
  - remove v from the network and correct capacities of the neighboring vertices
- `return` b

”Three Indians” Algorithm

The run of the algorithm is presented on the animation.

Zasd_ilustr_q.swf

”Three Indians” Algorithm

Note that we have chosen the vertex that has the smallest capacity, in the procedures `FORWARD` and `BACKWARD` we can always send the surplus or compensate the insufficiency.

The main loop will be executed at most $n - 2$ times, because each time we saturate one vertex in the graph.

Let us now count how many times we increase flow on edges in the procedures `FORWARD` and `BACKWARD`.

Edges will be saturated at most m times.

”Three Indians” Algorithm

We will increase the flow without saturating edges at most $O(n^2)$, because

- each time the the procedures `FORWARD` and `BACKWARD` are executed we send flow without saturating an edge at most once.
- there are at most $O(n^2)$ calls to these procedures.

Hence we need $O(n^2)$ time to find a blocking flow.

Together with with the blocking flow algorithm we get an algorithm finding maximum flow in $O(n^3)$ time.