COS 597B: Recent Advances in Graph Algorithms September 11, 2024

Lecture 3: Intro to Max Flow & Expander Decomposition

Lecturer: Huacheng Yu *Scriber: Pachara Sawettamalya*

In this lecture, we cover topics which will be needed later in the course. These include the problem of max-flow via Push-Relabel Algorithm, and expander graphs and expander decomposition.

1 Max Flows

Let $G = (V, E, c)$ be a weighted directed graph with two distinct vertices called *source* $s \in V$ and *sink* $t \neq s \in V$. Each directed edge $(u, v) \in E$ is also equipped with a positive capacity $c(u, v)$. For simplicity, we shall assume that s has no incoming edges, and neither does t has an outgoing edges.

Definition 1 (Flows and Maximum Flows) *Given a graph $G = (V, E, c)$. We call $f : E \to \mathbb{R}_{\geq 0}$ a* flow *iff the following conditions are satisfied.*

(1) capacity constraints: $f(e) \leq c_e$ for any $e \in E$.

(2) flow conservation: for any $v \in V \setminus \{s, t\}$, we must have $\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w)$. [1]

The value *of the flow f, denoted $val(f)$, is defined as*

$$val(f) := \sum_{(s,v)\in E} f(s, v) = \sum_{(u,t)\in E} f(u, t).$$

A flow with maximum value is called a maximum flow*.*

The max-flow problem is now well-defined: given a graph $G = (V, E, c)$, find its max flow. Perhaps the most well-known algorithms in the literature are the Ford-Fulkerson Algorithm ($O(mf)$ runtime where f is the value of max-flow), and the Edmond-Karp Algorithm ($O(mn^2)$ runtime). Both algorithms rely heavily on the idea of *residual graphs*.

Definition 2 (Residual Graph) *Given a graph $G = (V, E, c)$ and a flow f. We define a* residual graph *of G with respect to f, denoted $G_f = (V, E_f, c_f)$, to be a directed graph with the following set of edges:*

$$\forall (u, v) \in E, \text{ inserts to } E_f \text{ an edge} \begin{cases} (v, u) \text{ with capacity } f(u, v) \\ (u, v) \text{ with capacity } c(u, v) - f(u, v). \end{cases}$$

Note that by the definition above, it is possible for an edge (u, v) to be inserted twice into G_f (once via a flow through (u, v) and once via a flow through (v, u).) In this case, simply combine weights of the two edges into one.

Residual graphs are extremely useful objects to study maximum flows. Given a flow f (not necessarily maximum) of a graph G, we can construct a residual graph G_f. It turns out that if there exists a directed path from s to t in G_f, we can augment such flow into f, resulting in a larger flow. Conversely, we can show that a flow f is maximum if there are no such paths in G_f. These simple ideas motivate the essence of both Ford-Fulkerson and Edmond-Karp Algorithms: keep augmenting the flow until we no longer can.

Lemma 3 *Let f be a flow of a graph G. Then, f is a maximum flow if and only if there is no directed path from s to t in G_f. These paths, if existed, are call* augmenting paths*.*

In the next subsection, we will study another algorithm solving max-flow called the *push-relabel algorithm.*

[1] A natural interpretation of flow conservation is that for any non-terminal vertex, the amount *inflow* and the amount of *outflow* must be equal.

1.1 Push-Relabel Algorithm

The Push-Relabel Algorithm is an alternative way to solve the max-flow problem. At its core, the algorithm maintains and iteratively updates two parameters: preflow $f : E \to \mathbb{R}_{\geq 0}$ (i.e. a relaxation of flows where we only require inflow to be at least outflow [2]), and labels $\ell : V \to \mathbb{Z}_{\geq 0}$ (i.e. a topogical ordering of edges in residual graphs). In particular, we want to maintain the two set of invariants.

- Preflow. It must be true of all time that

$$\forall v \in V \setminus \{s, t\}, \sum_{(u,v)\in E} f(u, v) \geq \sum_{(v,w)\in E} f(v, w). \tag{1}$$

  We shall refer to the difference between inflow and outflow an an *excess* denoted by

$$e_f(v) = \sum_{(u,v)\in E} f(u, v) - \sum_{(v,w)\in E} f(v, w)$$

  which is always non-negative.

- Label. It must be true of all that that

$$\forall (u, v) \in E(G_f),\ \ell(u) \leq \ell(v) + 1. \tag{2}$$

1.1.1 The Algorithm

The algorithm consists of (1) an initialization phase where we hard-wire starting values of the preflow f and label ℓ, and (2) an iterative procedure where we constantly send flows through edges (via a *push*), or change labels of vertices (via a *relabel*).

Initialization. We initialize of f and ℓ as follows. It is easy to verify that such values satisfy both invariants 1 and 2.

$$f(u, v) = \begin{cases} c(s, v) & ; \text{if } u = s \\ 0 & ; \text{if } u \neq s \end{cases} \quad \text{and} \quad \ell(v) = \begin{cases} n & ; \text{if } v = s \\ 0 & ; \text{if } u \neq s \end{cases}$$

Iterative Procedures. The algorithm iteratively updates the preflow f and label ℓ. At each step, we identify a vertex u whose excess is strictly positive and attempt to either

- *push*(u): decrease u's excess by sending flows to its neighbor with a smaller label (if exists).

Algorithm 1 : *push*(u)
Input: graph $G = (V, E, c)$, preflow f, label ℓ, vertex u

1: let v be a vertex such that $(u, v) \in E_f$ and $\ell(u) \geq \ell(v) + 1$
2: send $\delta = \min\{e_f(u), c_f(u, v)\}$ units of flow through (u, v) by
   (i) update $f(u, v) \leftarrow f(u, v) + \delta$
   (ii) update the excesses $e_f(u) \leftarrow e_f(u) - \delta$ and $e_f(v) \leftarrow e_f(v) + \delta$

[2] Denote a value of a preflow f to be $\sum_{(w,t)\in E} f(w, t)$ and say a preflow f is maximum if it is a preflow of largest value. With this definition, we can show that the value of maximum preflow and the value of maximum flow are equal. Thus, it suffices to find a maximum preflow.

- *relabel*(u): if it is impossible to *push*, increase the label of u.

Algorithm 2 : *relabel*(u)
Input: graph $G = (V, E, c)$, preflow f, label ℓ, vertex u

1: set $\ell(u) = 1 + \min_{v:(u,v)\in E_f} \ell(v)$

Putting everything together, we can describe the full Push-Relabel Algorithm as follows.

Algorithm 3 : Push-Relabel Algorithm
Input: graph $G = (V, E, c)$

1: initialization: set $f(u,v) = \begin{cases} c(s,v) & ; \text{if } u = s \\ 0 & ; \text{if } u \neq s \end{cases}$ and $\ell(v) = \begin{cases} n & ; \text{if } v = s \\ 0 & ; \text{if } u \neq s \end{cases}$

2: **while** there exists a vertex u such that $e_f(u) > 0$ **do**
3: **if** there exists v such that $(u,v) \in E_f$ and $\ell(u) \geq \ell(v) + 1$ **then**
4: *push*(u)
5: **else**
6: *relabel*(u)
7: output f

1.1.2 Analysis

It remains to justify the correctness and runtime of the algorithm. We first argue that throughout the algorithm, the two inviriants hold.

- Preflow (1). Any calls to *relabel* do not alter the flow f. After calling a *push*(u) which sends $\delta = \min\{e_f(u), c_f(u,v)\}$ units of flow through (u,v), the excesses only change at two vertices u and v. For v, the excess increases by δ, thus remains positive. For u, the excess has becomes $e_f(u) - \delta$ which is still non-negative.
- Label (2). Any calls to *push* do not alter the label ℓ. Upon calling *relabel*(u), it must be case that $\ell(v) \geq \ell(u)$ for all $(u,v) \in E_f$. Then, u is relabelled to $1 + \min_{v:(u,v)\in E_f} \ell(v) \geq 1 + \ell(u)$. This means the label of u only increases. Consider three types of an edge $e \in E_f$.
  - (i) $e = (w,u)$ for $w \neq u$. After relabeling u, the label of u only increases. Thus, (2) still holds for e.
  - (ii) $e = (u,w)$ for $w \neq u$. After relabeling u, we have $\ell(u) = 1 + \min_{v:(u,v)\in E_f} \ell(v) \leq 1 + \ell(w)$. Thus, (2) still holds for e.
  - (iii) u is neither of the endpoints of e. In this case, the labels of e's endpoints are unchanged. Thus, (2) still holds for e.

Claim 4 *Suppose that Algorithm 3 terminates. Then, it outputs a maximum flow.*

Proof Let f be a flow upon the termination. By Lemma 3, f is a max-flow if and only if G_f has no augmenting path. Therefore, it suffices to show the latter.

Assume, for the contrary, that there exists an acyclic augmenting path $(s, v_1, ..., v_{k-1}, t)$ in G_f with $k \leq n-1$. Then, we have

$$\begin{aligned} \ell(s) \leq \ell(v_1) + 1 \leq \ell(v_2) + 2 \leq ... \leq \ell(v_{k-1}) + (k-1) &\leq \ell(t) + k && \text{(Label Invariant 2)} \\ &\leq \ell(t) + (n-1) && (k \leq n-1) \end{aligned}$$

Observe further that the labels of s and t are unchanged throughout the algorithm; thus we have $\ell(s) = n$ and $\ell(t) = 0$. This yields a contradiction. ■

Hence, our task remains to argue that G terminates, and does terminate with a fast runtime. It is in fact possible to show that G terminates in time $O(mn^2)$ for a weighted-capacity graph; however, to our interests we will show that when all edges have capacity 1, the algorithm runs in a much faster time. We first need a helper lemma.

Lemma 5 *Let f a flow at any point in the algorithm. Then, any vertex u such that $e_f(u) > 0$ must exists a path from u to s in the residual graph G_f. Note that we can further assume that the path is acyclic and has length at most $n - 1$.*

Proof Let G' be a subgraph of G consisting of edges with strictly positive flow in f. It suffices to show that there exists a directed path from s to u in G' (because its reversal is a path from u to s in G_f.) Let A be a set of all vertices *not* reachable from s in G'. Consider the following expression.

$$\begin{aligned} 0 \;\leq\; \sum_{v \in A} e_f(v) \;&=\; \sum_{v \in A} \left(\sum_{(u,v) \in E(G')} f(u,v) - \sum_{(v,w) \in E(G')} f(v,w) \right) \\ &= \sum_{v \in A} \sum_{(u,v) \in E(G')} f(u,v) - \sum_{v \in A} \sum_{(v,w) \in E(G')} f(v,w) \end{aligned} \tag{3}$$

Furthermore,

$$\begin{aligned} \sum_{v \in A} \sum_{(u,v) \in E(G')} f(u,v) &= \sum_{\substack{(u,v) \in E(G') \\ u,v \in A}} f(u,v) + \sum_{\substack{(u,v) \in E(G') \\ u \notin A, v \in A}} f(u,v) \\ &= \sum_{\substack{(u,v) \in E(G') \\ u,v \in A}} f(u,v) \qquad \text{(no edges in } G' \text{ from } V \setminus A \text{ to } A) \end{aligned}$$

and

$$\sum_{v \in A} \sum_{(v,w) \in E(G')} f(v,w) = \sum_{\substack{(v,w) \in E(G') \\ v,w \in A}} f(v,w) + \sum_{\substack{(v,w) \in E(G') \\ v \in A, w \notin A}} f(v,w).$$

Thus, inequality (3) becomes:

$$0 \;\leq\; \sum_{v \in A} e_f(v) \;=\; \sum_{\substack{(v,w) \in E(G') \\ v \in A, w \notin A}} -f(v,w) \;\leq\; 0$$

which implies that $e_f(v) = 0$ for all $v \in A$. In other words, any vertex v *not* reachable from s in G' must have $e_f(v) = 0$. However, we know that $e_f(v) > 0$. This means v is reachable from s in G'. ■

Claim 6 *Suppose all of G's edges have capacity 1. Then, Algorithm 3 terminates in time $O(mn)$.*

Proof It suffices to show that the algorithm makes $O(mn)$ calls to *push* and *relabel*. Let f and ℓ be the flow and label at anytime throughout the algorithm. Consider a call to *relabel*(u). We must have have $e_f(u) > 0$ (via line 2). By Lemma 5, there exists an acyclic directed path $(u, v_1, ..., v_{k-1}, s)$ in G_f with $k \leq n - 1$. By the label invariant (2), we then have

$$\ell(u) \leq \ell(s) + k \leq n + (n - 1) = 2n - 1.$$

This means any vertex u is never relabeled more than $O(n)$ times. Thus, the number of calls to *relabel* is $O(n^2)$.

Now let's bound the number of *push*. Consider any (u, v) that can possibly appear in G_f. Because (u, v) has capacity at most two in G_f, we can *push* at most twice through (u, v) before it disappears from G_f. It is also necessary that $\ell(u) \geq \ell(v) + 1$ for the *push* to happen (via line 3).

If the *push*(es) causes (u, v) to disappear from G_f, its reverse edge (v, u) must appear instead. For (u, v) to reappear in G_f, we must send (at least) a *push* through (v, u), requiring that $\ell(v) \geq \ell(u) + 1$. For this to happen, we must *relabel* the vertex v. Otherwise $\ell(v)$ would have never changed while $\ell(u)$ might have only been increased; thus $\ell(v)$ would have still be at most $\ell(u) - 1$, which is a contradiction.

To sum up, once (u, v) appears in G_f, we can send at most two *push*es through it, and the edge will disappear. To be able to send more *push*es through (u, v), we must *relabel* v at least once. Thus, the number of *push* through (u, v) is upper-bounded by the $O(\ell(v))$, which is $O(n)$. There can be at most $O(m)$ possible edges in G_f. Thus, the total number of pushes is $O(mn)$. ■

2 Expanders and Expander Decomposition

Before diving expanders, let us first agree on the notations. Let $G = (V, E)$ be an unweighted undirected graph. For any $S, T \subseteq V$, we denote

$$E(S, T) = \{(s, t) \in E \ ; s \in S, t \in T\}$$

to be the set of edges between S and T. Also, for a subgraph H of G and $S \subseteq V(H)$, denote the *volume* of S with respect to H by

$$vol_H(S) = \sum_{s \in S} deg_H(s).$$

When the context is clear, we might drop the subscript and refer to $vol_G(S)$ as $vol(S)$.

The core idea of expanders is , although sounded counter-intuitive, the it is possible for a graph to be tightly-connected yet sparse. We first formalize one possible measure of graph connectivity via the definition of *conductance*.

Definition 7 (Conductance) *Let $(S, V \setminus S)$ be a non-empty cut of G. We define a* conductance *of S to be*

$$\Phi_G(S) := \frac{|E(S, V \setminus S)|}{\max\{vol(S), vol(V \setminus S)\}}.$$

Then, the conductance of G is defined to be

$$\Phi_G := \min_{\substack{S \subseteq V \\ S \notin \{\emptyset, V\}}} \Phi_G(S).$$

Definition 8 (Expander Graphs) *Say a graph G is ϕ-expander if and only if $\Phi_G \geq \phi$.*

Let's take a step back and digest the definition of conductance. Had we ignored the denominator $\max\{vol(S), vol(V \setminus S)\}$ of $\Phi_G(S)$, the definition simply becomes that of a min-cut. Intuitively, a graph with a large min-cut can be interpreted as being highly connected. However, it is likely that a min-cut of G is very unbalanced ($|S| \ll |V \setminus S|$) which mean that the connectivity of the graph has been undermined by the smallness of S. Therefore, using min-cut as a measure of graph connectivity might not be the most accurate. Here via the notion of conductance, each cut size is normalized by $\max\{vol(S), vol(V \setminus S)\}$, which is a proxy to the balance-ness of both sides of a cut $(S, V \setminus S)$. Therefore, we can interpret a graph with high conductance as a graph with a large and balanced min-cut.

It turns out that an arbitrary graph G might have a low conductance. Surprisingly, G can be partitioned into multiple parts of high conductance with only a small number of edges between different parts.

Theorem 9 (Expander Decomposition) *Let $G = (V, E)$ be a graph with n vertices and m edges. There exists a partition of V into $\mathcal{U} = \{U_1, ..., U_k\}$ such that*

(1) $\sum_{i=1}^{k} |e_G(U_i, V \setminus U_i)| \leq \tilde{O}(\phi m)$ [3]

(2) $G[U_i]$ *is a ϕ-expander for all $i \in [k]$* [4]

It is worth noting that there is a surprising connection between the expander decomposition and the low-diameter decomposition we saw in the previous lectures. Consider setting $\phi = 1/D$, and the decomposition of V into $\mathcal{U} = \{U_1, ..., U_k\}$. By (1), the number of removed edges is bounded by $\tilde{O}(m/D)$. By (2), each induced subgraph $G[U_i]$ is a ϕ-expander. It is also known that a ϕ-expander graph has diameter $\tilde{O}(\phi^{-1})$. Thus, each induced subgraph $G[U_i]$ has diameter $\tilde{O}(D)$.

2.1 Proof of Theorem 9

In this subsection, we proof the Expander Decomposition Theorem. We claim that the following algorithm produces a decomposition satisfying both conditions. On a high-level, the algorithm can be described as "iteratively splitting a non-ϕ-expander part into two smaller parts."

Algorithm 4 : Expander Decomposition
Input: a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, an expansion rate ϕ
Output: $\mathcal{U}$ which is a ϕ-decomposition of G

1: initialize $\mathcal{U} \leftarrow V$
2: **while** there exists $U \in \mathcal{U}$ such that $G[U]$ is not a ϕ-expander **do**
3: let a non-empty set $S \subset U$ be such that $vol_{G[U]}(S) \leq vol_{G[U]}(U \setminus S)$ and $\frac{|E(S, U \setminus S)|}{vol_{G[U]}(S)} < \phi$
4: replace, in $\mathcal{U}$, the set U with $(S, U \setminus S)$
5: output $\mathcal{U}$

Trivially, Algorithm 4 terminates because we can decompose the partition at most n times. Also clear (by construction) is that upon termination, the condition (2) is satisfied. Therefore it remains to bound the number of deleted edges. This is shown via the following claim.

Claim 10 *The number of deleted edges is $\tilde{O}(\phi m)$.*

Proof Consider an arbitrary round where we replace the set U with $(S, U \setminus S)$. We charge each vertex $v \in S$ with amount $\phi \cdot vol_G(v)$. Thus, the amount being charged to S is

$$\sum_{v \in S} \phi \cdot vol_G(v) = \phi \cdot vol_G(S) \geq \phi \cdot vol_{G[U]}(S) > |E(S, U \setminus S)| \tag{4}$$

For any vertex v, denote $\psi(v)$ to be the total amount of charge to v throughout the algorithm. By (4), the number of removed edges is bounded above by $\sum_{v \in V} \psi(v)$.

Now consider arbitrary vertex v. For every time that v is charged, the volume of the partition containing v is down by at least a half (via line 3). Such volume starts off at most n^2, and ends up

[3] $\tilde{O}(f) = f \cdot polylog(n)$
[4] $G[U]$ is an induced subgraph on U.

at least 1. Thus, v can be charged at most $2\log_2 n$ times. Moreover, each charge is of the amount $\phi \cdot vol_G(v)$. Thus, we have

$$\begin{aligned}(\#\text{ removed edges}) \;\leq\; \sum_{v\in V} \psi(v) \;&\leq\; \sum_{v\in V} 2\log_2 n \cdot (\phi \cdot vol_G(v)) \\ &=\; 4\phi m \log_2 n \\ &=\; \tilde{O}(\phi m)\end{aligned}$$

as wished. ■