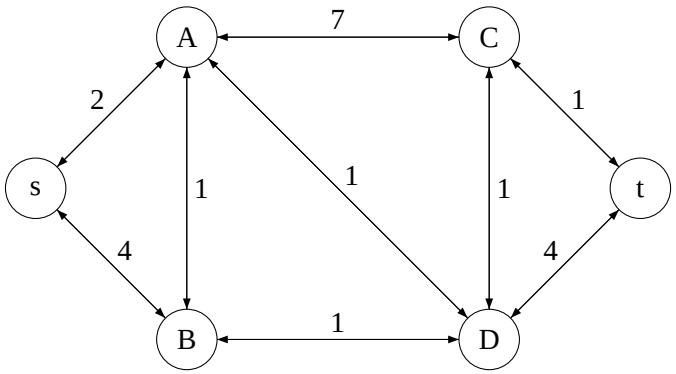
This lecture describes both sequential and parallel versions of a maximum flow algorithm based on the preflow-push method.

16.1 Maximum Flow Definitions

Before describing the preflow-push algorithm, we'll first establish some definitions.

Definition 16.1.1 *A **flow network** is a directed graph $G = (V, E)$ with a source $s \in V$, a sink $t \in V$, and capacities along each edge (described by a function $c : E \to \mathbb{R}$ where $c(e)$ is the capacity of edge e).*

Example: Consider the following flow network G.



■

Definition 16.1.2 *The amount of **flow** between two vertices is described by a function $f : V \times V \to \mathbb{R}$. The flow function f has the following properties:*

***Capacity**: $f(v, w) \leq c(v, w) \quad \forall v, w \in V$*

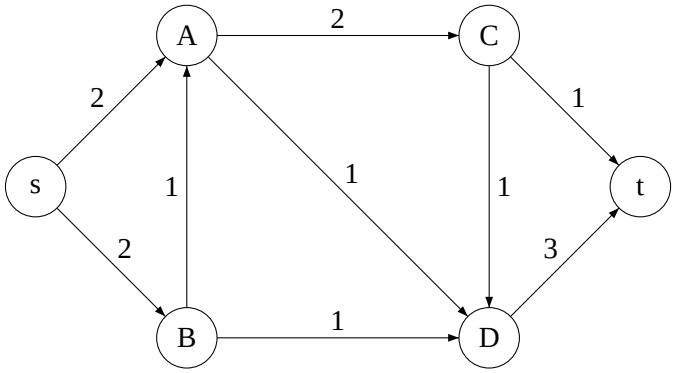
***Antisymmetry**: $f(v, w) = -f(w, v) \quad \forall v, w \in V$*

***Conservation**: $\sum_{w \in V} f(v, w) = 0 \quad \forall v \in V - \{s, t\}$*

Definition 16.1.3 *The **total flow** $|f|$ of a flow network is the amount of flow going into the sink. Formally, $|f| = \sum_{v \in V} f(v, t)$.*

We're interested in finding the **maximum flow**, the largest possible $|f|$ for a given graph G.

Example: The maximum flow of the flow network G is 4.



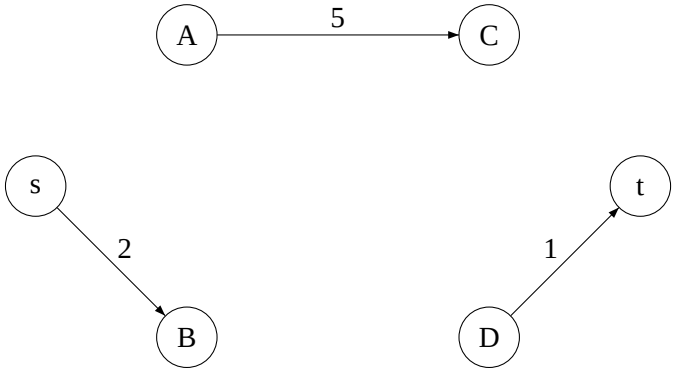
Note that only the positive flows between vertices are shown above (all flows ≤ 0 are omitted). ■

Definition 16.1.4 *The **residual** across two vertices $v, w \in V$ is described by function $r : V \times V \to \mathbb{R}$ such that $r(v, w) = c(v, w) - f(v, w)$. Thus, the residual $r(v, w)$ represents the amount of potential flow we can still push from v to w.*

Definition 16.1.5 *The set of **residual edges** E_R consists of all vertex pairs with positive residuals. Formally, $E_R = \{(v, w) \in V \times V \mid r(v, w) > 0\}$.*

Property 16.1.6 *A total flow $|f|$ is maximum $\Leftrightarrow$ s and t are disconnected in $G_R = (V, E_R)$.*

Example: The following is the graph G_R after the maximum flow has been pushed across the flow network G.



Note that s and t are disconnected because we've pushed the maximum flow across this flow network. ■

16.2 Pre-flow Push Overview and Definitions

Now that we've got proper definitions for the maximum flow problem, we can produce an algorithm to find the maximum flow of a graph using the preflow-push method.

The basic idea of the preflow-push method is to relax the conservation property of the flow function. Specifically, in intermediate steps of the algorithm, we'll allow vertices to have more flow coming in than going out.

To do this we'll redefine f to be the preflow (instead of flow) function. Preflow f maintains the capacity and antisymmetry flow constraints, but we replace the conservation constraint with a nonnegativity constraint: $\sum_{w \in V} f(w, v) \geq 0 \quad \forall v \in V - \{s\}$.

After the termination of the preflow-push algorithm, we'll see that the final preflow f must satisfy the conservation constraint (meaning that it is a valid flow) and must be the maximum flow $|f|$.

Definition 16.2.1 *The **excess** of a node $v \in V$ is defined by the function $e : V \to \mathbb{R}$ with $e(v) = \sum_{w \in V} f(w, v)$. Because of the nonnegativity constraint of the preflow f, we know that $e(v) \geq 0$.*

Using this definition of excess, we take a node $v \in V - \{s, t\}$ to be **active** if and only if $e(v) > 0$.

Definition 16.2.2 *A **valid labeling** is a function $d : V \to \mathbb{Z}$ with the following properties:*

i. $d(s) = n$

ii. $d(t) = 0$

iii. $d(v) \leq d(w) + 1 \quad \forall (v, w) \in E_R$

Informally, $d(v)$ is called the "height" of $v \in V$. The source starts as the highest point in the graph, while the sink is the lowest.

16.3 Pre-flow Push Sequential Algorithm

The preflow-push algorithm relies on two underlying operators:

Push: For an active $v \in V$ where there exists a $w \in V$ with $r(v, w) > 0$ and $d(v) = d(w) + 1$, we push $min(e(v), r(v, w))$ flow along the edge (v, w).

Relabel: For an active $v \in V$, where for all other $w \in V$, $r(v, w) > 0 \Rightarrow d(v) \leq d(w)$, we can relabel $d(v) = min\{d(w) + 1 : (v, w) \in E_R\}$.

The sequential algorithm is simply the following:

```
// initialize preflow
foreach (v, w) ∈ (V − {s}) × (V − {s}) do
    f(v,w) = 0;
    f(w,v) = 0;
end
foreach v ∈ V do
    f(s,v) = c(s,v);
    f(v,s) = -c(s,v);
end
// initialize labels and excesses
d(s) = n;
foreach v ∈ V − {s} do
    d(v) = 0;
    e(v) = f(s,v);
end
// perform operations
while there exists an active vertex do
    perform an applicable operation;
end
return f;
```

Claim 16.3.1 *As long as there is an active vertex, we can always perform a push or relabel operation.*

Claim 16.3.2 *The push and relabel operations maintain the preflow and valid labeling properties.*

Lemma 16.3.3 *For any valid labeling d, the source s and the sink t must be disconnected in G_R.*

Proof: Assume for sake of contradiction that there exists a path $v_0, v_1, \ldots, v_k$ in G_R with $v_0 = s, v_k = t$. Each vertex is only used once in the path, meaning that $k < n$. Also, $(v_i, v_{i+1}) \in E_R$ for all $0 \leq i < k$. By definition of a valid labeling, we know that $d(v_i) \leq d(v_{i+1}) + 1$ for all $0 \leq i < k$. We can deduce then that $d(s) \leq d(t) + k$.

$d(s) \leq d(t) + k$

$\Rightarrow n \leq 0 + k$ (because $d(s) = n, d(t) = 0$).

$\Rightarrow n \leq k$.

But we've already shown that $k < n$. Contradiction.

Therefore, s and t cannot be connected in G_R. ■

Corollary 16.3.4 *If there are no more active vertices, then the current preflow is the maximum flow.*

Proof: If there are no more active vertices, then $e(v) = 0 \quad \forall v \in V - \{s, t\}$. Thus, by the definition of $e(v)$, this implies that $\sum_{w \in V} f(v, w) = 0 \quad \forall v \in V - \{s, t\}$. This preflow then satisfies the flow conservation constraint, meaning that f is a valid flow. Furthermore, since we have a valid labeling d of the vertices, Lemma 15.3.3 tells us that s and t are disconnected in G_R. By Property 15.1.6, this means that $|f|$ is maximum. ■

16.4 Parallel Preflow-Push Algorithm

The following is a rough sketch of the parallel version of the preflow-push algorithm (for a more rigorous definition, look to page 395 of the "Preflow Push Max Flow" handout on the course website):

initialize preflows, excesses, and;
while *there exists an active vertex* **do**
  (1) **forall** *active v* **do** push as much flow over v given current e(v);
  (2) **forall** *active v* **do** relabel(v) if possible;
end

Steps 1 and 2 of each round can be performed in combined work $O(m)$ and depth $O(logn)$. We need to provide an upper bound on the number of rounds executed before the algorithm terminates. Before providing this bound, we'll establish some useful claims and properties.

Claim 16.4.1 *At any point during the algorithm,* $d(v) \leq 2n - 1$ *for all* $v \in V$.

Proof: By definition of a valid labeling, the claim is trivial for $v = s$ and $v = t$.

Since the preflow-push algorithm only modifies the labels of active vertices, all we need to show is that the claim holds for any active vertex v.

Suppose $v \in V - \{s, t\}$ with v active. Thus, v has positive excess. This implies there is a simple path from v to s in G_R (for proof of this, see Lemma 3.5 of the "Preflow Push Max Flow" handout). Take this path to be $v_0, v_1, \ldots, v_k$ with $v_0 = v, v_k = s$. Since this is a simple path in G_R, $k \leq n - 1$. Since d is a valid labeling and $(v_i, v_{i+1}) \in E_R$, we know that $d(v_i) \leq d(v_{i+1}) + 1$. From this, we can deduce that $d(v) = d(v_0) \leq d(v_k) + k \leq d(s) + (n - 1) = 2n - 1$. Therefore, $d(v) \leq 2n - 1$. ■

Property 16.4.2 *For all vertices* $v \in V$, $d(v)$ *never decreases over the course of the preflow-push algorithm. Therefore, by Claim 15.4.1, the total number of relabels is* $\leq (n - 2)(2n - 1) \leq 2n^2$.

With these claims and properties, we're able to prove bounds on the number of rounds before termination of this algorithm.

Theorem 16.4.3 *The number of rounds executed before the termination of this algorithm is* $O(n^2)$.

Proof: Take $\Phi = max\{d(v) : v \text{ active}\}$. Take $L = \sum_{v \in V} d(v)$.

For each round of the algorithm, we have three cases:

Case 1: There are no label changes. This only happens if each vertex moves all of its excess to lower labeled vertices. Thus, for each vertex v with the maximum $d(v) = \Phi$, v must have pushed its entire excess $e(v)$ to some neighbor w with $d(v) = d(w) + 1$ (by the conditions necessary for push). Additionally, no vertex can push any flow to v in this iteration because v is the highest vertex. Since each of these vertices with the current maximum height pushed all of their excess, these vertices are no longer active. This implies that the maximum height Φ must decrease from its current value. Specifically, Φ decreases by 1 because each of the neighbors w that received flow from a previously highest active vertex v must now be active (because it has received positive excess from v) and must satisfy $d(v) = d(w) + 1$ (by the conditions necessary for the push to occur).

Case 2: Some relabeling occured, but Φ stays the same. This means that some vertex v was relabeled.

Each relabel operation strictly increase the height of the node being relabeled. Thus, L increases by at least 1.

Case 3: Some relabeling occured, and Φ increases by k. Thus, since the maximum is now at least k larger than the previous maximum, L must also increase by at least k.

We've shown by Property 15.4.2 that this algorithm can perform at most $O(n^2)$ relabel operations. Therefore, cases 2 and 3 can only occur $O(n^2)$ times.

Since cases 2 and 3 can only occur $O(n^2)$ times, this means that Φ can stay the same or increase in only $O(n^2)$ rounds. At the beginning of the algorithm, $\Phi = 0$ because all $v \in V - \{s, t\}$ have height 0 (and $d(s) = n$ but s is never considered active). At the termination of the algorithm, $\Phi = 0$ because no nodes are active. Since Φ can increase at most $O(n^2)$ rounds of the algorithm (and the initial and end values of Φ are 0), then Φ must also only decrease in at most $O(n^2)$ rounds. Thus, case 1 can also only occur $O(n^2)$ times.

Since all of the cases are bounded by $O(n^2)$ iterations, we can conclude that there are at most $O(n^2)$ rounds before the termination of the algorithm. ■

Therefore, since each round can be performed in $O(m)$ work and $O(logn)$ depth, the parallel preflow push algorithm has overall work $W(n) = O(n^2m)$ and depth $D(n) = O(n^2logn)$.

16.5 Closing Remarks

While many graph algorithms have been successfully parallelized, there still are certain algorithms with no significant parallel counterparts. For instance, the "single source shortest path" problem (solved sequentially by Dijkstra's algorithm) currently has no solution with significant parallelism.