

Push-Relabel and Bipartite Matching

Advanced Algorithms (CSE 794)

Instructor: Tamal K. Dey

Scribe: Ted Bulwinkle *

April 30, 2009

1 Push-Relabel

Consider a flow network $G = (V, E)$ where $n = |V|$, $m = |E|$. The Push-Relabel algorithm explained here is Goldberg's generic maximum-flow algorithm, which runs in $O(n^2m)$ time, an improvement on the $O(nm^2)$ time of the Edmonds-Karp algorithm.

1.1 Preliminaries

Instead of maintaining the flow conservation property, we maintain a preflow f that is anti-symmetric and satisfies the capacity constraints, and allow excess flow $e(u) = f(V, u)$ at the vertices.

We maintain a height function to control how flow is pushed through the network:

Height: $h(u) \leq h(v) + 1$ if $(u, v) \in E_p$, where E_p is the residual network.

The height of the source s is fixed at $|V|$ and the height of the sink is fixed at 0. All other vertices start with height 0, and are increased as the algorithm progresses, and flow is only allowed from a higher vertex to a lower vertex.

If there is an edge from u to v in the residual network, then we are going to push excess flow from u to v if the height difference is exactly 1, subject to capacity constraints. That is, v is lower than u , but just by 1.

PUSH(u, v):

Conditions: $e(u) > 0$, $c(u, v) > 0$, $h(u) = h(v) + 1$

Action: Push $d_f(u, v) = \min(e(u), c_f(u, v))$ through (u, v)

$$e(u) = e(u) - d_f(u, v)$$

$$e(v) = e(v) + d_f(u, v)$$

It is a saturating push if $c_f(u, v) = 0$ afterward, ie. if we push the maximum possible amount. It is a non-saturating push if $e(u)$ was the amount pushed.

If u is lower than all surrounding vertices, then we cannot Push. In particular, when we have a vertex u with excess, but the height of u is less than or equal to all residual edges, then we "relabel" the height of u to 1 above the minimum height v . Then we can Push after the Relabel.

*Department of Computer and Information Sciences, The Ohio State University, Columbus, Ohio, USA.

RELABEL(u):

Condition: $e(u) > 0, s.t. with (u, v) \in E_p, h(u) \geq h(v) \forall v$

Action: $h(u) = 1 + \min\{h(v) : (u, v) \in E_f\}$

INITIALIZE():

$h(s) = |V| = n$, source is initially the number of v .

$h(t) = 0$

All edges of the form (s, u) have

$c_f(s, u) = 0$,

$e(u) = c(s, u)$, all the edges going out of s have an excess flow at u .

$e(s) = -\sum_u c(s, u)$ thus the source vertex has a negative flow.

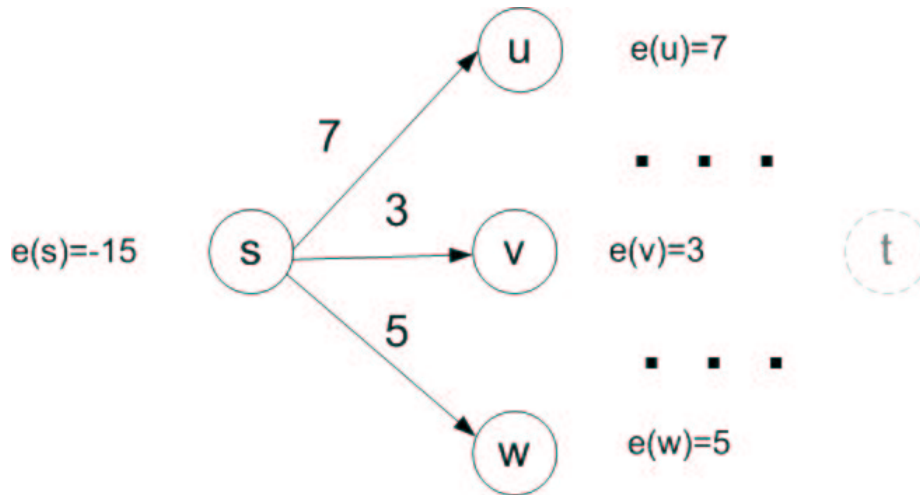


Figure 1: Excess flow at source node s and adjacent vertices after Initialize. Non-pictured nodes and sink node t are initialized with excess flow 0.

1.2 Algorithm

PUSH-RELABEL(G)

1. Initialize;
2. While there exists an applicable Push or Relabel apply it.

The idea is that we are draining the excess flow from the source, and at the time there is no excess flow, it is a flow. Furthermore we'll show it's a maximum flow.

Lemma 1. For any u, v in V , if $h(u) < h(v) + 1$ then (u, v) is not in E_f

This follows directly from the height function.

Lemma 2. After a non-saturating push from u to v $e(u) = 0$

This is obvious from the Push operation. Whatever excess flow we had on u , we have let it go.

Lemma 3. *An overflowing vertex can either be pushed or relabeled.*

Proof. If Push is not applicable, $h(u) < h(v) + 1 \Rightarrow h(u) \leq h(v)$ so we can Relabel, since f is a preflow.

Lemma 4. *Vertex heights never decrease.*

Proof. Relabel only increases the height, and push doesn't change it. By the preconditions on u in Relabel, $h(u) \geq h(v) \forall v.s.t.(u, v) \in E_f$. Thus, Relabel increases the height of u by at least 1.

Lemma 5. *The "height function" property is maintained throughout the algorithm.*

Proof. We need to show that if h is a height function, then h is a height function after both the Relabel and Push operations.

Relabel: on a residual edge (u, v) Relabel(u) ensures $h(u) \leq h(v) + 1$ after the Relabel operation; likewise on a residual edge (v, u) Relabel(u) ensures $h(v) < h(u) + 1$ (by Lemma 4).

Push: Consider Push(u, v). If it adds (v, u) to E_f , then $h(v) = h(u) - 1$. If it removes edge (u, v) , from the residual network, the height constraint is also removed.

Lemma 6. *Suppose f is a preflow, h is a height function on V . Then there is no path from s to t in G_f*

Proof by contradiction. Suppose there was such a path. Then there can be at most v vertices in this path. This height can decrease at most $v-1$ (num of edges in a simple path). But s is at height $|V|$, and t is at 0 by definition. Thus the path cannot exist.

Theorem. *If Push-Relabel(G) terminates, then preflow f is a flow and furthermore is a maximum flow for G .*

Proof. If it terminates, we can't Push or Relabel, i.e. each vertex in $V - s, t$ has an excess flow of 0. By Lemma 3, there are no overflowing vertices, so preflow f is a flow. By Lemma 6, this flow can't be augmented, so it must be a maximum flow by the max-flow min-cut Theorem.

Lemma 7. *For any overflowing vertex u , there exists a simple path from u to s in the residual network G_f .*

Proof by contradiction. The idea is every time you Push something, you maintain a path back to the source. Assume that there isn't a simple path from s to v in G_f . Let $U = \{v: \text{there exists a simple path from } u \text{ to } v \text{ in } G_f\}$. Let $U' = V - U$. Then $\forall v \in U$ and $\forall w \in U', f(w, v) \leq 0$ since if $f(v, w) < 0$ then $c_f(v, w) = c(v, w) - f(v, w) > 0$. But this implies there is an edge $(v, w) \in E_f$, which would contradict the fact that $w \in U'$ since we can construct the path $\{u, \dots, v, w\}$. Therefore, $f(U', U) \leq 0$ and $e(U) = f(V, U) = f(U', U) + f(U, U) = f(U', U) \leq 0$. However, this then implies that $e(u) = 0$, which contradicts the assumption that u is overflowing, so therefore there must exist a simple path from u to s .

The remaining lemmas serve to prove algorithm termination, which will give us the upper bound on the time complexity. We will show there are finite Relabel, saturating Push, and non-saturating Push operations.

Lemma 8. *At any time, we have $h(u) \leq 2n - 1, u \in V - s, t$*

Proof. This follows from Lemma 7, that height can't increase forever. By Lemma 7, there is a simple path p from u to s in G_f . For each vertex $v_i \in p$, we have $(v_i, v_{i+1}) \in E_f$. By Lemma 5, $h(v_i) \leq h(v_{i+1}) + 1$. Expanding over p , we have $h(u) \leq h(s) + (n - 1) = 2n - 1$.

Lemma 9. *The number of Relabel operations is at most $2n-1$ per vertex and at most $(2n-1)(n-2) < 2n^2$ overall.*

Proof. This follows from Lemma 8 since Relabel changes the height by at least 1. Consider $u \in V - s, t$. By Lemma 8, $h(u)$ grows to at most $2n-1$. There are $n-2$ of these vertices, so the total number of operations is at most $(2n-1)(n-2) < 2n^2$.

Lemma 10. *There are at most $2nm$ saturating pushes.*

Consider an edge (u,v) . Suppose there is a saturating push from u to v . Then the edge disappears. In order for the edge to reappear, there must be a flow from v to u , ie. we have to push back at some point. So inbetween 2 saturating pushes from u to v , the height of v must increase by at least 2, since the height of u was 1 more than v , and the other way v was 1 more than u .

Lemma 11. *There are at most $4n^2(n+m)$ non-saturating pushes.*

Proof. Define a potential function $\Phi = \sum_v (h(v))$, where $e(v) > 0, \forall v$. (Ie. we are thus summing the heights of overflowing vertices only.) Now we consider what happens when Φ increases or decreases.

Φ Increases:

Relabel: Obvious, it increases height. But we can't increase Φ by more than $2n-1$ by Lemma 8 and there are at most $2n^2$ relabel ops by Lemma 9, for a total of $4n^3$

Saturating Push: Something can leave the height sum, but something can come in $< 2n-1$ times. This can happen at most $2nm$ times for a total $< 4n^2m$.

Φ Decreases:

Non-saturating push: Decreases by 1.

$e(u)$ becomes 0, so $h(u)$ goes out of the sum

$e(v)$ goes in, but by push pre-condition $h(u)=h(v)+1$

so Φ decreases by 1.

Thus, the total amount of increase in Φ is $\leq 4n^3 + 4n^2m = 4n^2(n+m)$ and the total amount of decrease in Φ is constrained by this amount, so therefore the total number of non-saturating pushes is at most $4n^2(n+m)$.

It follow directly that the total cost of the algorithm is $O(n^2m)$ which is at most $O(n^4)$, an improvement over the Edmonds-Karp algorithm which runs in $O(nm^2)$ which is at most $O(n^5)$.

2 Bipartite Matching

G is an undirected graph $G=(V,E)$ a matching M is M subset E , and any vertex v in V has at most 1 edge in M incident to it.

A bipartite matching is the same, but with the constraint that G is a bipartite graph. See Figure 2.

A maximum matching is a matching M s.t. \forall matchings $M', |M| \geq |M'|$.

Goal: find a maximum bipartite matching. This is also known as the marriage problem: consider men to be in the left subset of V , and women to be on the right subset of V , in a bipartite graph G . Then the goal is to maximize the number of marriages, s.t. a man isn't married to more than one woman, (and vice-versa).

It turns out that we can solve this as a maximum flow problem.

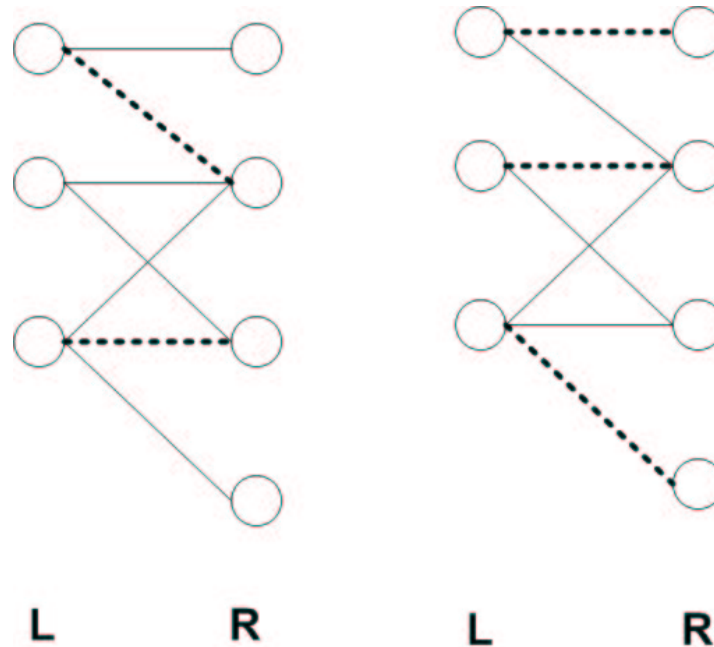


Figure 2: A bipartite Graph G with vertex partition $V = L \cup R$. (**Left**) Dotted edges show a matching M of cardinality 2. (**Right**) Dotted edges show a maximum matching of cardinality 3. Note that there are multiple maximum matchings.

How do we solve this as a maximum flow problem? Add a source and sink to the bipartite graph, and make the graph directional from source to sink. See Figure 3.

Define a flow f to be integer valued if for any (u, v) , $f(u, v)$ is an integer.

Lemma 12. Suppose $G(V, E)$ is a bipartite graph with $V = L \cup R$ and all edges go between L and R and $G' = (V', E')$ is the flow network. If M is a matching in G , then there is an integer flow f in G' with $|f| = |M|$ and conversely if f is an integer valued flow in G' , then there is a matching M in G with cardinality $|M| = |f|$.

Proof. First show that a matching in G corresponds to an integer valued flow in G' . Define f as: if $(u, v) \in M$, then $f(s, u) = f(u, v) = f(v, t) = 1$ and $f(u, s) = f(v, u) = f(t, v) = -1$; otherwise $f(u, v) = 0$. Then each edge $(u, v) \in M$ corresponds to 1 unit of flow in G' along the path s, u, v, t . So, f satisfies skew symmetry, capacity constraints, and flow conservation, since f can be obtained by flow augmentation along each path. The net flow across cut $(L \cup s, R \cup t) = |M|$ and thus the net flow is $|f| = |M|$.

Conversely, let f be integer valued in G' and let $M = (u, v) : u \in L, v \in R, \text{ and } f(u, v) > 0$. Each vertex u has only one centering edge (s, u) of capacity 1. Thus 1 unit of positive net flow enters u if and only if there is exactly 1 vertex v s.t. $f(u, v) = 1$. Thus at most one edge leaving each u carries positive net flow. A symmetric argument can be made for each v . Thus M is a matching. Then $|M| = f(L, R) = f(L, V') - f(L, L) - f(L, s) - f(L, t) = 0 - 0 + f(s, L) - 0 = f(s, V') = |f|$.

Theorem. If the capacity c takes only integral values, then max-flow f produced by Ford-Fulkerson algorithm has the property that f is integral.

Proof is by induction on each iteration of the Ford-Fulkerson algorithm. Since the capacity c takes only integral values, the flow is only going to change by an integral amount at each step of the algorithm.

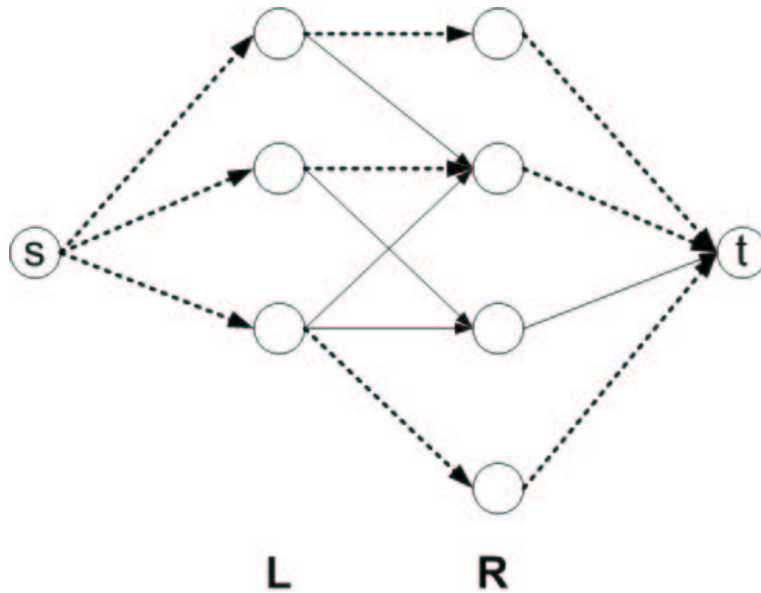


Figure 3: The flow network corresponding to the bipartite graph from Figure 2. Each edge has unit capacity; dotted edges have a flow of 1; solid edges have no flow. The dotted edges from L to R correspond to a maximum matching.

Corollary *Furthermore, the cardinality of a maximum matching in a bipartite graph G is the value of a maximum flow in its corresponding flow network G'*

Proof by contradiction. Suppose M is a maximum matching in G and that the corresponding flow in G' is not a maximum flow. Then there is a f' in G' s.t. $|f'| > |f|$. Since the capacities in G' are integer valued, so is f' . Thus, f' corresponds to a matching M' s.t. $|M'| = |f'| > |f| = |M|$. This contradicts our assumption that M was a maximum matching. There is a symmetric argument to show that if f is a maximum flow in G' , then it corresponds to a maximum matching on G .

The running time of this algorithm is $O(|E||f^*|) = O(m|f^*|)$, since we can find a maximum matching by running Floyd-Fulkerson and directly extracting the maximum matching M . Since the size of the flow $|f^*|$ is at most the number of edges, the size of M is $O(n)$ by the nature of the bipartite graphs.

3 References

Dey, Tamal K.. Lecture for CSE 794: Advanced Algorithms, Department of Computer and Information Sciences, The Ohio State University, Columbus, Ohio, USA. April 7, 2009.

Cormen, Leiserson, and Rivest. Introduction to Algorithms. The MIT Press. 1990.