Bipartite Matching and Push-Relabel

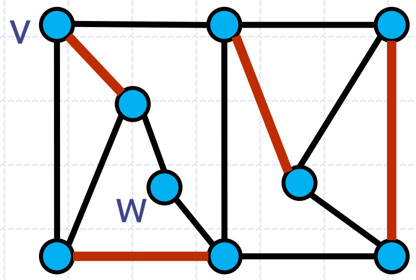
Chapter 26

Lecture Overview

- Maximum Bipartite Matching
  - Solved using Ford-Fulkerson
- Push-relabel Method
  - Push Operation
  - Relabel Operation
  - Algorithm
  - Correctness
  - Analysis

Maximum Matching

Given a graph G = (V, E), a matching in G is a subset M of E such that no vertex is incident on more than one edge.

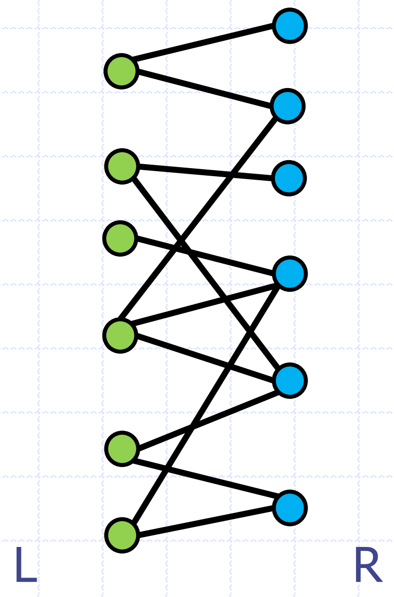


A vertex is matched by the matching if it is incident on an edge, and unmatched otherwise.

A maximum matching is a matching of maximum cardinality. We are interested in the problem of finding a maximum matching for a graph. However, we will restrict our attention to bipartite graphs.

Bipartite Graphs

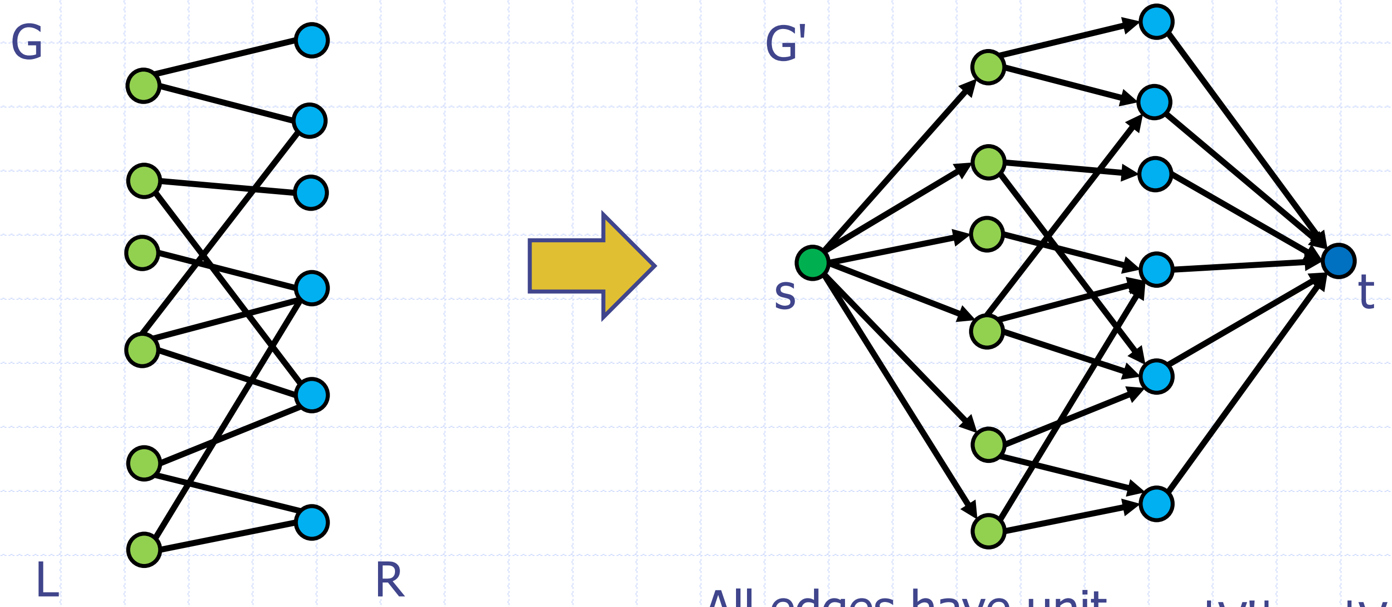
A graph is bipartite if its vertices can be partitioned into two sets L and R such that every edge of the graph goes between one vertex in L and one vertex in R.



The problem of finding a maximum matching in a bipartite graph has many applications. For instance, we may have a set L of machines and a set R of tasks, and an edge between l and r means that machine l can perform task r. In this setting, a maximum matching is an assignment of tasks to machines that maximizes the number of tasks being worked on.

Maximum Bipartite Matching

- We shall construct maximum bipartite matchings by converting the bipartite graph to a flow network and running Ford-Fulkerson on it.

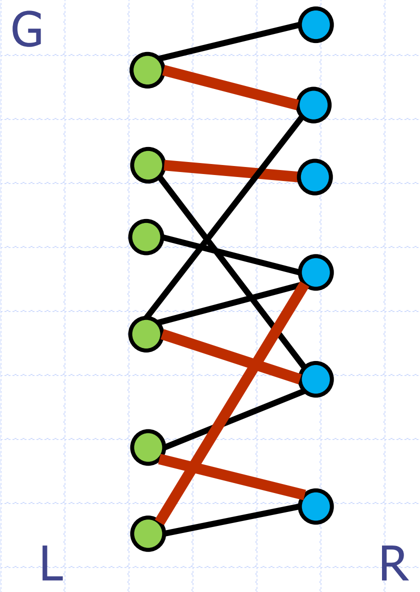


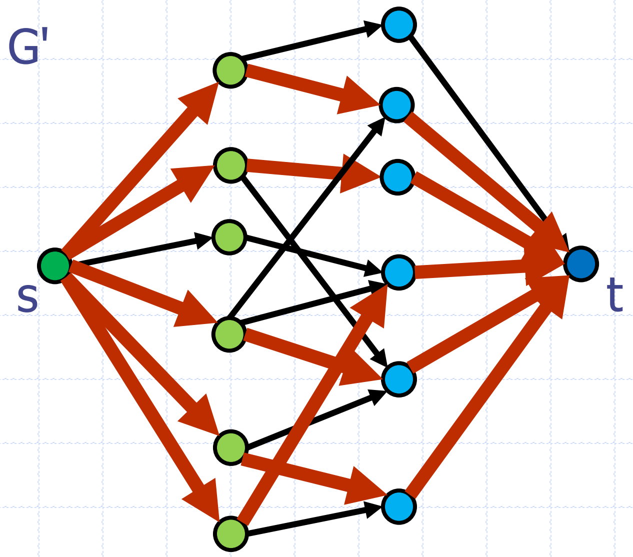
All edges have unit capacity

|V'| = |V| + 2
|E'| = |E| + |V|
≤ 3 |E|

Maximum Bipartite Matching

Lemma. Let G = (V, E) be a bipartite graph with partition V = L ∪ R and G' be the corresponding flow network. If M is a matching in G, there is an integer-valued flow f in G' with value |f| = |M|. Conversely, if f is an integer-valued flow in G', then there is a matching M in G with cardinality |M| = |f|.





Integrality Theorem for Ford-Fulkerson

The lemma almost says that we can run Ford-Fulkerson on G', get a flow, and from it derive a matching on G. Unfortunately, the lemma talks about integer flows and not general flows.

It turns out that we can run Ford-Fulkerson on G', because Ford-Fulkerson actually guarantees that the flows are integer:

Theorem. If the capacities are all integers, then the maximum flow f produced by Ford-Fulkerson has the property that for all vertices u and v, f(u, v) is an integer.

Integrality Theorem for Ford-Fulkerson

Proof (sketch). Prove by induction on the iterations of Ford-Fulkerson. At each step, the residual network has only integral capacities. Therefore, the augmenting path flow is integral. Sums of integers are integers. ■

So to find a matching in undirected bipartite graph G:

1. form directed flow network G'
2. run Ford-Fulkerson on it
3. the matching is the edges of G with flow 1

Maximum Bipartite Matching

Step 1: takes $O(V + E)$ time.

Step 2: Any matching has $O(V)$ edges, so the maximum flow f* will have value $O(V)$. Our analysis told us that Ford-Fulkerson runs in time $O(|f^*|E')$, which in this case is $O(VE)$.

Step 3: takes $O(V)$ time (as that is the size of the matching)

Therefore, in total, the algorithm for Maximum Bipartite Matching takes time $O(VE)$.

Push-Relabel

- Push-relabel is another method of solving network flow problems—and not just maximum flow problems. This method is generally faster than Ford-Fulkerson.
- We examine Goldberg's maximum-flow algorithm. It runs in $O(V^2E)$ time, which is better than Edmonds-Karp's $O(VE^2)$ time. It can be improved to $O(V^3)$ time, as shown in the text (section 26.5), but we will not cover that.
- Push-relabel algorithms look at one vertex at a time and its neighbors in the residual graph, rather than the whole residual graph as in Ford-Fulkerson.

Preflows and Overflowing

Push-relabel methods also do not maintain flow-conservation throughout the algorithm. They instead maintain a preflow f, which satisfies the capacity constraint and

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

for all vertices u in V – {s}. We call e(u) the excess flow into vertex u.

We call a vertex u overflowing if e(u) > 0.

Metaphor

Directed edges will correspond to pipes. Vertices are pipe junctions, have two properties

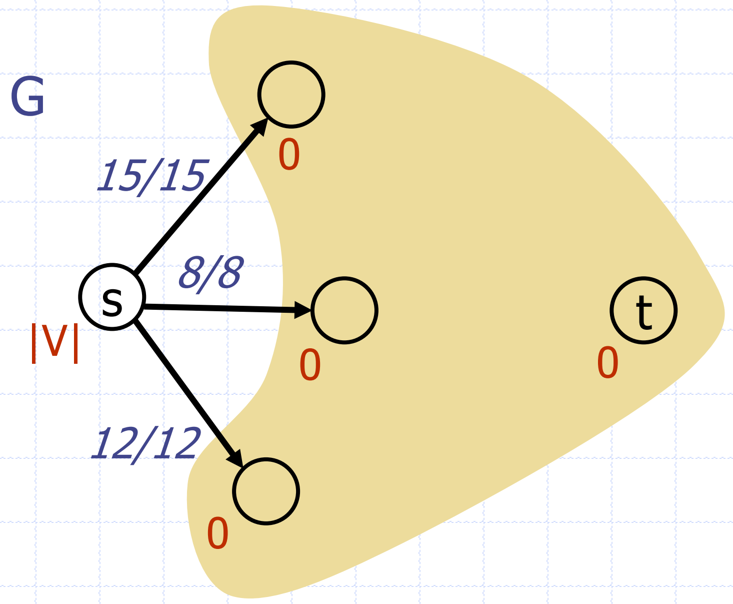
1. to accommodate excess flow, each vertex has an outflow pipe to an arbitrarily large resevoir.

2. each vertex has a height which can increase as the algorithm progresses.

Vertex heights determine determine how flow is pushed; we only push flow downhill. The source and sink are fixed at heights |V| and 0, respectively. All other vertices start at 0 and increase with time.

Metaphor

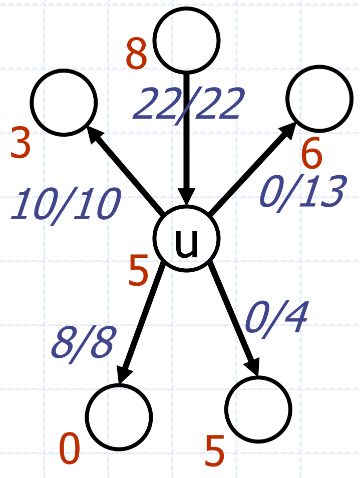
The first step of the algorithm is to send as much flow as possible downhill from the source, saturating each outgoing edge.



When flow first enters an intermediate vertex, it collects in the vertex's reservoir. Eventually it is pushed downhill.

Metaphor

The algorithm may discover that the only pipes that leave a vertex u and are not already saturated with flow are not downhill from u.



If u is overflowing, then we may increase its height by an operation called relabeling. We increase its height to one more than the height of the lowest neighbor to which it has an unsaturated pipe.

After relabeling u, we can push more flow from u.

Metaphor

Eventually, all flow that can possibly get through to the sink has arrived there. To make the preflow into a valid flow, we send the excess collected in the reservoirs of overflowing vertices back to the source by continuing to relabel and push.

Formally, let f be a preflow in flow network G. A height function h has $h(s) = |V|$, $h(t) = 0$, and $h(u) \leq h(v) + 1$ for every residual edge (u, v) in E_f.

Lemma. If $h(u) > h(v) + 1$, then (u, v) is not an edge of the residual network.

The Push Operation

The operation PUSH(u, v) applies if u is an overflowing vertex, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

```
PUSH(u, v)
1. change = min(u.excess, $c_f(u, v)$)
2. **if** (u, v) $\in$ E
3.     (u, v).flow = (u, v).flow + change
4. **else**
5.     (v, u).flow = (v, u).flow – change
6. u.excess = u.excess – change
7. v.excess = v.excess + change
```

The Push Operation

A push is saturating if the edge (u, v) in the residual network becomes saturated (has $c_f(u, v) = 0$ afterwards). If an edge becomes saturated, it disappears from the residual network.

Observation. After a nonsaturating push from u to v, the vertex u is no longer overflowing.

Observation. If f is a preflow before the push operation, then f is a preflow after the push operation.

The Relabel Operation

The operation RELABEL(u) applies if u is overflowing and u.height $\leq$ v.height for all edges (u, v) $\in E_f$.

```
RELABEL(u)
1. u.height = $1 + \min\{v.height : (u, v) \in E_f\}$
```

Since we only call RELABEL when u is overflowing, there is at least one edge (u, v) in the residual network, so the min is over at least one item.

Initialize-Preflow

```
INITIALIZE-PREFLOW(G, s)
1.  for each vertex v in G.V
2.      v.height = 0
3.      v.excess = 0
4.  for each edge (u, v) in G.E
5.      (u, v).flow = 0
6.  s.height = |G.V|
7.  for each vertex v in s.Adj
8.      (s, v).flow = c(s, v)
9.      v.excess = c(s, v)
10.     s.excess = s.excess – c(s, v)
```

Generic Push-Relabel

```
GENERIC-PUSH-RELABEL(G)
1. INITIALIZE-PREFLOW(G, s)
2. while there is an applicable push or relabel operation
3.     select such an operation and perform it
```

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

Corollary. When the algorithm terminates, the preflow f is a flow.

No s-t Path Lemma

During the operation of Generic Push-Relabel, the vertex heights never decrease. Also, the heights maintain the formal properties of a height function.

Lemma. Let G = (V, E) be a flow network with source s and sink t, let f be a preflow in G, and h be a height function on V. Then there is no path from the source s to the sink t in the residual network G_f.

Proof. Assume there is a simple path p = $<v_0, v_1, \ldots, v_k>$ where $v_0 = s$ and $v_k = t$. $k < |V|$. For i = 0...k-1, edge (v_i, v_{i+1}) is in E_f. Because h is a height function, $h(v_i) \leq h(v_{i+1}) + 1$ for i = 0...k-1. Combining all of these inequalities yields $h(s) \leq h(t) + k$, a contradiction.

Push-Relabel Correctness

Theorem. If GENERIC-PUSH-RELABEL terminates when run on a flow network G, then the preflow it computes is a maximum flow for G.

Proof. We prove the following invariant:

At each iteration of the **while** loop of GENERIC-PUSH-RELABEL, f is a preflow.

INITIALIZATION: INITIALIZE-PREFLOW makes f a preflow.
MAINTENANCE: The only operations performed are PUSH and RELABEL. RELABEL does not affect f. A previous observation is that a PUSH changes a preflow into another preflow.

Push-Relabel Correctness

TERMINATION: The preflow f must be a flow as all excesses are 0. The heights are a height function, so there is no path from s to t in the residual network G_f. By the Max-flow Min-cut Theorem, then, f is a maximum flow. ■

The analysis of GENERIC-PUSH-RELABEL is involved. Please read it. The major idea is to bound the number of operations: relabels, saturating pushes, and nonsaturating pushes. First we bound the heights:

Lemma. For all vertices u in V, u.height never exceeds $2|V|-1$.
Corollary. The number of relabel operations is less than $2|V|^2$.

Push-Relabel Analysis

Lemma. The number of saturating pushes is less than 2|V||E|.

Proof (sketch). Count the saturating pushes on (u, v) and (v, u) together. As pushes are downhill, max(u.height, v.height) gets one bigger each saturating push. Therefore, at most 2|V| saturating pushes on (u, v) and (v, u). ■

Lemma. The number of nonsaturating pushes is less than 4|V|^2(|V| + |E|).

Proof. By potential method. In text. ■

Push-Relabel Analysis

Theorem. The number of basic operations in GENERIC-PUSH-RELABEL on any flow network $G=(V, E)$ is $O(V^2E)$.

Corollary. There is an implementation of the generic push-relabel algorithm that runs in $O(V^2E)$ time.