Matching

A matching in an undirected graph is a set of edges, no two having a common end vertex.

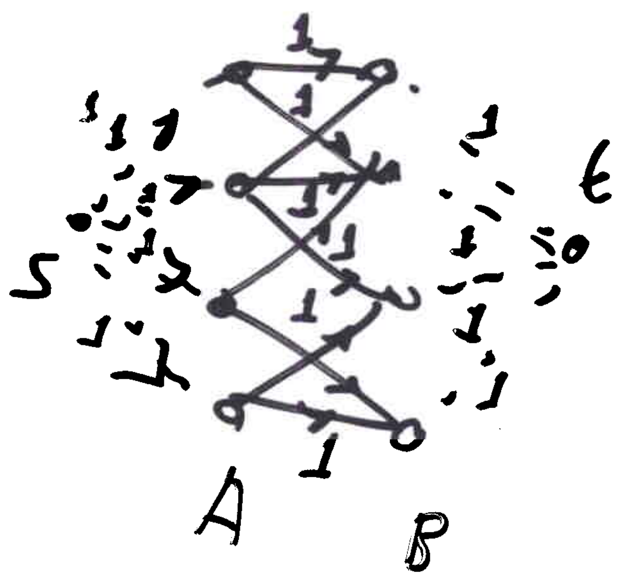
Bipartite graph: vertices can be partitioned into two sets, such that every edge has one end vertex in each set.

Maximum cardinality matching: find a matching containing as many edges as possible.

Maximum weight matching: in a graph with edge weights, find a matching with maximum total weight.

Bipartite vs. general graphs

Bipartite!!



Augmenting Paths

free free

one more matched edge

— = unmatched

~~ = matched

M = any matching $\bar{M}$ = max (card.) matching

$M \oplus \bar{M}$ = edges in exactly one of $M, \bar{M}$:

subgraph, all degrees ≤ 2:

— $\bar{M}$

~~ M

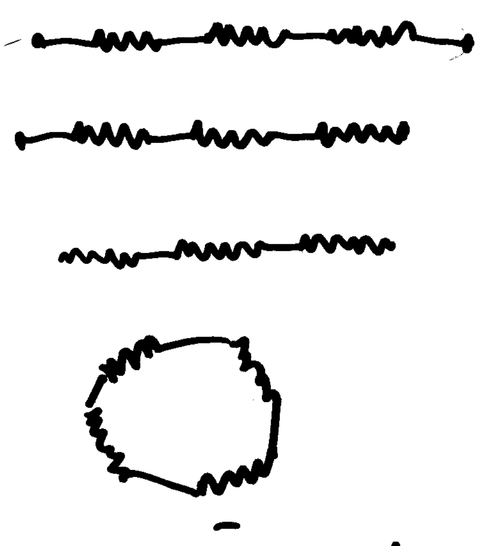
$\bar{M}$ excess

+1 M-augmenting

0

-1

0



If $|\bar{M}| - |M| = k$, $M \oplus \bar{M}$ contains k M-aug. paths

Max card matching

Begin with empty matching.

Repeatedly find an augmenting path, augment.

Stop when no more augmenting paths.

Bipartite case:

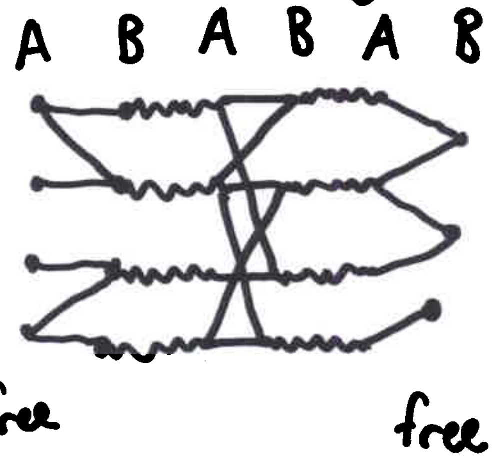
$O(m)$ time per augmentation.

$O(n)$ augmentations

$\Rightarrow O(nm)$ time total.

Bipartite case faster

Build layered subgraph containing all shortest aug paths by BFS



S

Find aug paths in S 1 at a time by DFS

Total time per phase = $O(m)$.

Length of shortest aug path strictly increases after a phase

$O(\sqrt{n})$ phases $\Rightarrow$ $O(\sqrt{n}\, m)$ time

Each phase increases aug path length:

Let $d(v)$ be shortest dist from an A-free vertex to v via an alternating path.

$d(v)$'s strictly increase along any shortest aug. path. New edges created by a shortest aug. go from larger to smaller $d(v)$.

Thus no shorter aug path created by a shortest aug; after a phase, every aug path contains at least one edge from larger to smaller $d(v) \Rightarrow$ longer path.

$2\sqrt{n}$ phases:

Each phase increases matching size.

If $|\bar{M}| - |M| > \sqrt{n}$, $M \oplus \bar{M}$ contains $> \sqrt{n}$ aug paths, at least one of length $< \sqrt{n}$ (only n vertices).

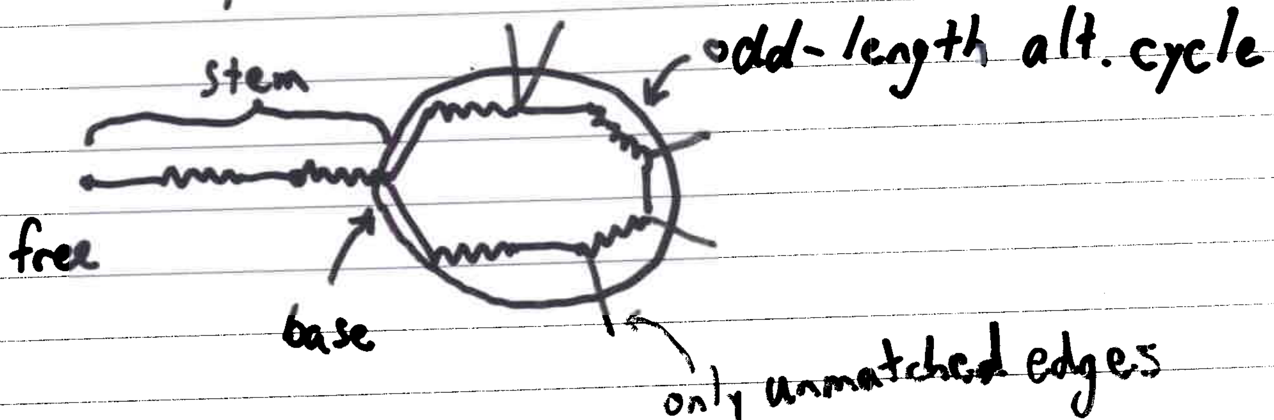
$\Rightarrow$ After $\sqrt{n}$ phases, shortest aug path has length $\geq \sqrt{n}$ $\Rightarrow$ within $\sqrt{n}$ of max $\Rightarrow$ $\leq \sqrt{n}$ more phases.

Max card matching on general graphs

Basic problem: how to find one aug path

(a vertex can be an A-vertex or a B-vertex; a priori, one doesn't know which)

Edmonds: blossom-shrinking to find aug paths

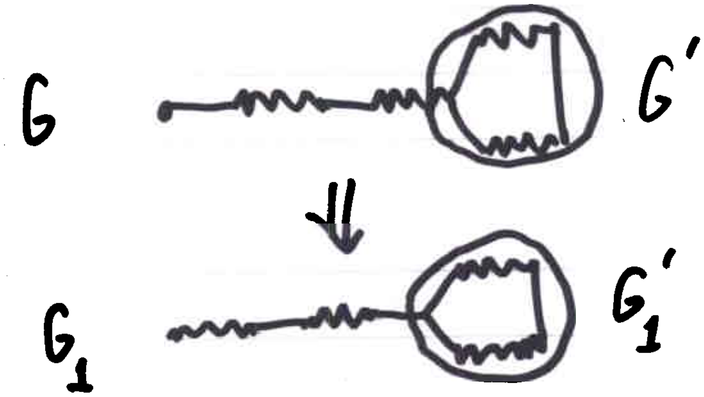


Thm: Let G' be formed from G by shrinking a blossom. Then G' contains an aug path iff G does.

Pf. If G' contains an aug path, then G does: expand blossom, link broken ends of path by going around blossom in correct direction (one broken end is blossom base).

Other direction is the hard part.

If the blossom has a non-trivial stem, swap edges along it to make the base of the blossom free, obtaining G_1 from G (and G_1' from G').



G (G') has an aug path iff G_1 (G_1') does.

Thus we need only show that if G_1 has an aug path, so does G_1'. Thus suppose G_1 has an aug path. Either it is an aug path in G_1' or it hits the blossom, in which case the part from ~~the~~ end not the blossom base until it first hits the blossom is an aug path in G_1'.

Edmonds' alg to find an aug path
via blossom-shrinking (DFS version)

Start at any free vertex.

Grow an alt. search path.

If an edge extending the path hits the path,
shrink a blossom if the path is of odd length;
otherwise discard the edge.

When reaching a new free vertex, stop with
success.

When at a vertex or blossom with no unexplored
edges, delete the vertex or blossom.

After deleting a free vertex, start a new search
at an undeleted free vertex.

Time per aug path: $O(m\alpha(n))$

(need set union to maintain blossoms)

Total time $= O(nm\alpha(n))$

Can improve to take advantage of

shortest aug path idea:

<u>very</u> complicated

Nothing better is known, even though sum of lengths of shortest aug paths is $O(n \log n)$.

Note: k phases $\Rightarrow$ max to within $(1 - 1/k)$ factor: fast approximation

Generalizes to general graphs, weighted matchings, shortest paths, max flows

$O(\sqrt{n}\, m) \times \alpha$ and/or log factors

	bipartite	general
cardinality	Hopcroft & Karp, 1971 $O(n^{1/2}m)$	Micali & Vazirani, 1980 $O(n^{1/2}m)$
weighted	Fredman & Tarjan, 1984 $O(n^2\log n + nm)$ Gabow, 1985 $O(n^{3/4} m \log C)$ Gabow & Tarjan, 1987 $O(n^{1/2} m \log (nC))$	Gabow, Galil, & Spencer, 1984 $O(n^2\log n + nm \log\log\log_{m/n} n)$ Gabow, 1985 $O(n^{3/4} m \log C)$ Gabow & Tarjan, 1987 $O((n\alpha(m,n)\log n)^{1/2} m \log (nC))$

Related Work

The cost scaling approach gives a time of $O(\sqrt{n}\, m \log(nC))$ for the assignment problem (weighted bipartite matching).

Compare with Hopcroft-Karp bound of $O(\sqrt{n}\, m)$ for unweighted bipartite matching, and Fredman-Tarjan bound of $O(nm + n^2 \log n)$ for a nonscaling algorithm

For nonbipartite weighted matching, we obtain a time of

$O(\sqrt{n\alpha(n,n) \log n}\; m \log(nC))$

Compare with Micali-Vazirani bound of $O(\sqrt{n}\, m)$ for unweighted matching, Gabow-Galil-Spencer bound of $O(nm \log\log\log_{m/n} n + n^2 \log n)$ for a nonscaling algorithm