15-451 Algorithms
Spring 2013
D. Sleator

Matchings February 12, 2013

I have borrowed words and ideas from these notes:
http://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Handouts/tarjan-blossom.pdf

Also see: http://en.wikipedia.org/wiki/Blossom_algorithm

0: Introduction

A matching in an undirected graph is a set of edges such that none of them have any endpoints in common. Today we're going to focus on the problem of computing a matching that has as many edges as possible.

Matchings have many applications. For example, a good approximation algorithm known for the travelling salesman problem involves computing a minimum cost matching in a graph. It's also a fundamental computational problem on graphs, and computing it is an interesting algorithmic challenge in its own right.

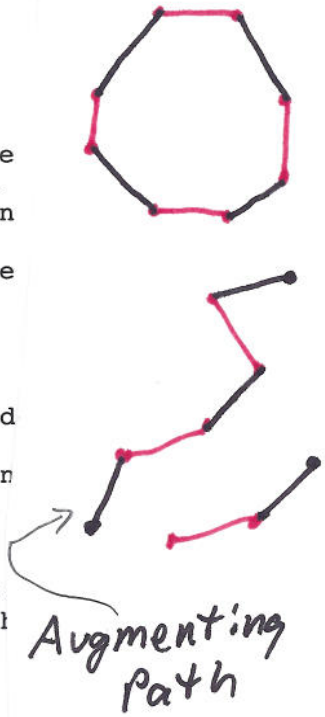
Computing the maximum matching in a bipartite graph is considerably simpler. It will come up again when we do network flow. Today we'll focus on algorithms that work for general graphs.

1: Augmenting Paths

Our general approach will be to build up the matching incrementally. At each step we will have some matching, which we will try to improve. When we get to a point where we cannot improve it, we're done.

Suppose we have a matching that is not maximum? Then what can we search for to try to improve it?

Let G=(V,E) be our graph. And let R be our current matching. Suppose there's another matching B which is bigger than R. Consider P, which is the set of edges in R or B, but not both. Let's color the edges in B blue, and the edges in R red, and look at the connected components of P. The degree of the vertices incident on edges of P are of degree 1 or 2. Therefore these components of P must be paths or cycles. (Those are the only kind of graphs where all the degrees are 1 or 2.) A cycle in P must be even in length, and contain the same number of red and blue edges. The paths could be of even length of or of odd length. The even length paths also contain the same number of red and blue edges. Because B is a bigger matching than R, it follows that there must be an odd length path that contains one more blue edge than red. This path begins with a blue edge, and alternates colors, and ends with a blue edge. This is called an augmenting path.



So going back to the original graph G and our current matching R, the augmenting path (which the above paragraph shows must exist) is a path in G which alternates between unmatched and matched edges. It is of odd length and starts and ends with an unmatched edge.

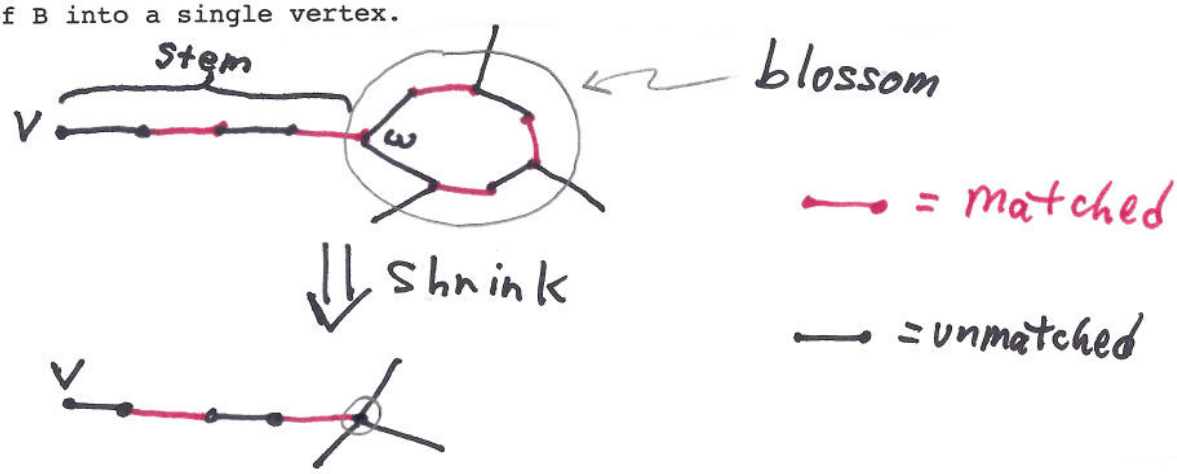
Given an alternating path, we can use it to increase the size of our current matching by toggling all the edges on it in and out of the matching.

So our approach is going to be to search for an augmenting path. If none exists, then we have a maximum matching. If one exists, we apply it (increasing the size of our current matching), and then continue.

---

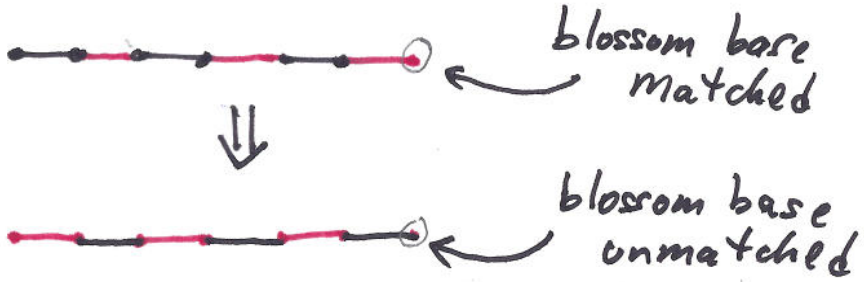
2. Blossoms

Consider an alternating even-length path P from a free (unmatched) vertex v to a vertex w plus an odd-length alternating cycle from w to itself. Cycle B is a blossom; path P is a stem; vertex w is the base of the blossom. Shrinking the blossom consists of contracting all vertices of B into a single vertex.



Theorem: Let G′ be formed from G by shrinking a blossom B. Then G′ contains an augmenting path iff G does.

Proof: If the base w of the blossom B is not free, change the matching in G by switching the edges along the stem to make w free (see figure below). Then B is a blossom with a free vertex as a base. Let G1 be G after this change and G1′ the graph resulting from shrinking B in G1. (G and G1, and also G′ and G1′, differ only in which edges are matched and which are unmatched). The switching does not change the matching size; thus G has an augmenting path iff G1 does, and G′ has iff G1′ does. Thus we need consider only the case in which the base of the blossom is free.



Suppose G′ has an augmenting path. Either this path is an augmenting path in G, or it ends at the blossom, and it can be extended to an augmenting path in G by following the blossom in the direction that results in alternation until reaching the base.

Suppose G has an augmenting path. Either it is an augmenting path in G′ or it hits the blossom, in which case the part from one end until the blossom is first hit is an augmenting path in G′.

QED.

---

3. Edmonds' Blossom Shrinking Algorithm

The reason we introduced blossoms is that we're going to give an algorithm that, given a graph G and a matching M (which is non-maximal), the algorithm either finds an augmenting path or it finds a blossom.

Why is this useful? Because if we find an augmenting path, we can grow the matching and continue. If we find a blossom, we can (1) shrink it, (2) recursively find an augmenting path in the shrunken graph and (3) extend that alternating path to the graph with the blossom restored. (Unlike the proof above, this last step needs to handle the case when the blossom base is not matched. This is easy to do. If the path goes through the blossom, it must use both a matched and an unmatched edge incident on the blossom. It is easy to extend such a path through the expanded blossom.)

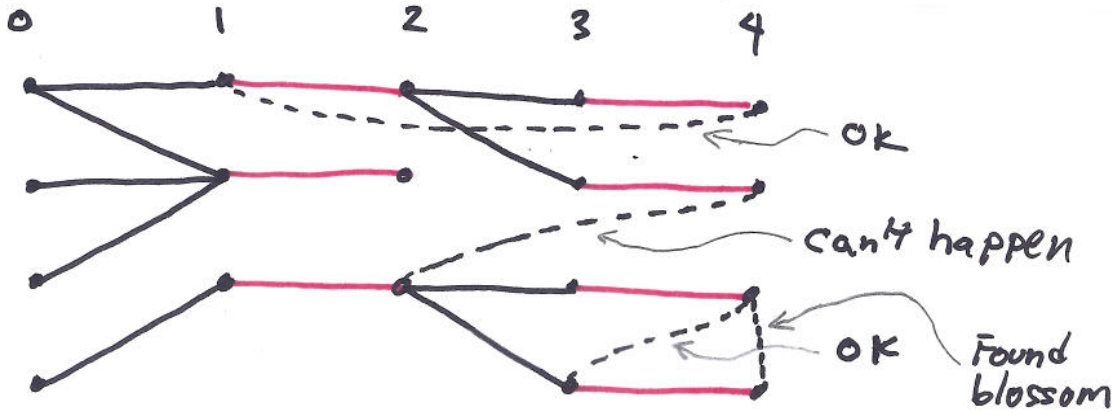
If our algorithm cannot find a blossom or augmenting path, then we will supply a proof that we already have the maximum matching.

So it sufficies to give an algorithm that takes as input a graph G and a sum-optimum matching M, and produces an augmenting path or a blossom.

Edmonds' Algorithm:

We do breadth first search in the graph. We construct a series of numbered layers. Layer 0 consists of all the free (unmatched) vertices. The vertices in layer i are neighbors of those in layer i-1. Each vertex keeps track of it's "parent" vertex in the previous layer caused it to be put there. Even and odd layers are constructed differently.

> Layer i (for i odd): Consider all non-matched edges incident to vertices in layer (i-1). The vertices reachable from these that have not yet been examined form layer i. If in the process we find an unmatched edge from layer (i-1) to layer (i-1) we have found a blossom or augmenting path.
>
> Layer i (for i even): Consider all matched edges incident to vertices in layer (i-1). If i the process we find a matched edge from layer (i-1) to layer (i-1) we have found a blossom or augmenting path.



The edges from an even layer to the next layer are all unmatched edges. The edges from an odd layer to the next layer are all matched. When examining the vertices on an even layer to construct the subsequent odd layer the edges used must be unmatched edges. This is because the even layer vertices are either unmatched, or were put into their layer by a matched edges, which are therefore accounted for.

Note that when we find the edge within a layer causing a blossom or augmenting path, we can easily construct that path by following pointers back from the corresponding vertices to layer 0. If the two paths meet, then we found a blossom, if they don't then we found an augmenting path.

Theorem: If the current matching has an augmenting path, then the algorithm will reach one of the states "found a blossom or augmenting path".

Proof: Consider the completeled layered construction given by the algorithm. Let's take the augmenting path and follow it along in the layered construction. Eventually we have to come to an edge that stays within the same level. This edge will therefore trigger the condition that we claim must happen.

Here's why this is true. If, as we traverse along the augmenting path, we change levels, we must also change the parity of the level we're on. If we're following a matched edge we go from one level to the next. If we're following an unmatched edge, that means we're on an even level. We can go back to a previous level, but not to another even level. Because there cannot be an unmatched edge connecting two even levels. (We would have explored the right endpoint of that edge when working on the earlier even level.)

The augmenting path is of odd length and starts and ends on level 0. Therefore there must be two neighboring points on the path with the same parity. This proves that the claimed condition must occur at some point in running the algorithm.

---

4. Running time

The graph has n vertices and m edges. The maximum matching is of size at most n/2. Therefore we augment the matching at most n/2 times. Each time we augment requires finding and shrinking at most O(n) blossoms. Each search for a blossom costs O(m) time. So the whole algorithm is O(n^2 m).

---

5. Tutte matrix

Given an undirected graph G, construct the following matrix A

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \text{ is an edge of } G \text{ and } i<j \\ -x_{ij} & \text{if } (i,j) \text{ is an edge of } G \text{ and } i>j \\ 0 & \text{otherwise} \end{cases}$$

If we view the xijs as variables, then the determinant of A is non-zero (as a polynomial) if and only if the graph has a perfect matching. (A perfect matching on an n node graph (n even) has n/2 matched edges.)

By assigning random values to the xijs and then computing the determinant, we get a probabilistic algorithm for determining if a graph has a perfect matching. This can also be used to construct the perfect matching. The running time, however, ends up being O(n^5) which is slower than the Edmonds algorithm. But it's simpler to program.

This is an example of the application of the Schwartz-Zippel lemma. We may come back to this later in the semester.