

Matchings II

December 11, 2003

1 Maximum Size Matching in a Non-Bipartite Graph

The results from the previous lecture suggests a very natural algorithm for computing a maximum size matching. Start from an empty matching M and repeatedly find an augmenting path from an unmatched vertex to an unmatched vertex which is an augmenting path. This can be done by collapsing the unmatched vertex to a single vertex s , and let H be the resulting graph.

Doing a BFS on H starting from s such that on even levels you must traverse the matching edges, and odd levels must traverse the unmatched edges.

An augmenting path in G corresponds to an odd cycle in H with s .

Consider the alternating BFS tree T we computed for H .

Lemma 1.1 *If there is an augmenting path in G for a matching M , then there exists two vertices u and v of the same depth in T , such that $uv \in E(G)$. Furthermore, if the depth of u and v in T is even, then uv is a free edge, if the depth of u and v in T is odd, then $uv \in M$.*

Proof: Let π be an augmenting path in G . The path π corresponds to a an odd length alternating cycle in H . Let σ be the shortest odd length alternating cycle in G (note that both edges in σ that are adjacent to s are unmatched).

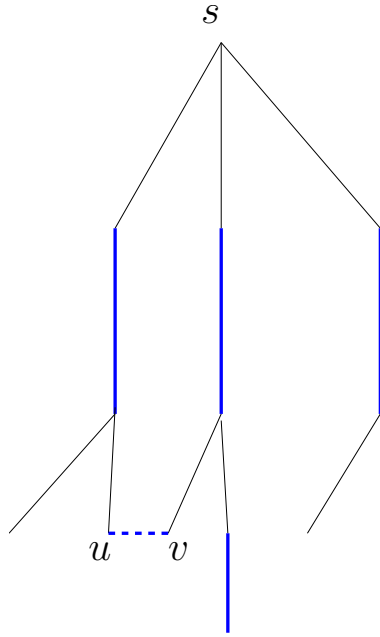
For every vertex x of σ let $d(x)$ be the length of the shortest alternating path between x and s in H . Similarly, let $d'(x)$ be the length of the shortest alternating path between s and x along σ . It is now easy to verify that $d(x) = d'(x)$ for all $x \in \sigma$, since otherwise there exists a shorter odd length alternating cycle with s on it.

But then, take the two vertices of σ 'furthest away from s . Clearly, both of them have the same depth in T , since $d(u) = d'(u) = d'(v) = d(v)$. Establishing the claim. See Figure 1. ■

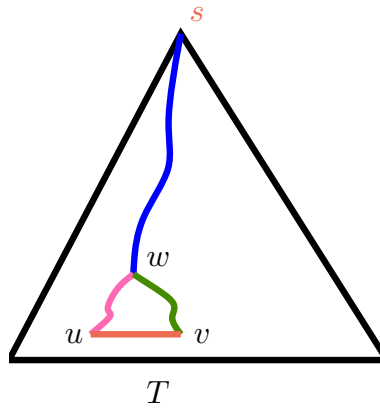
Thus, we can do the following: Compute the alternating BSP T for H . Find an edge uv of G which is connecting two vertices in the same level of T . If uv is an odd depth, it must be in the matching. If it is an even level, then it must be free (this automatically holds).

Extract the paths from s to u and from s to v in T , and glue them together with uv to an odd cycle μ in H . If μ corresponds to an alternating path in G then we are done, since we found an alternating path, we can apply it, and find a bigger matching.

But μ in fact, might have common edges. In particular, let π_u and π_v be the two paths from s to u and v respectively. Let w be the lowest vertex in T that is common to both π_u

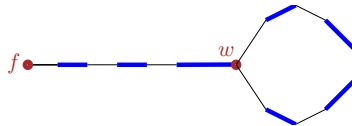


and π_v . Then μ is a blossom, namely, it is a path from s to w and odd length cycle based at w .



Let us translate this back to the original graph G . The path s to w corresponds to an alternating path starting at a free vertex f and ending at w , where the last edge is in the matching, the cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle where the two edges adjacent to w are matched.

This is look like:



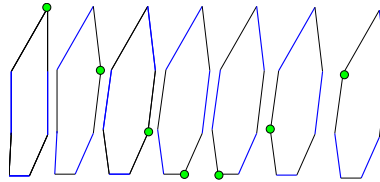
and we call it a **blossom**.

We can not apply a blossom to a matching in the hope of getting better matching. In fact, this is illegal and yield something which is not a matching. On the positive side, we discovered an odd alternating cycle in the graph G . Let us summarize this:

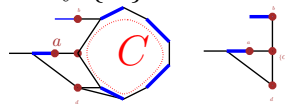
Lemma 1.2 *Given a graph G with n vertices and m edges, and a matching M , one can find in $O(n + m)$ time, either a blossom in G or an augmenting path in G .*

To see what to do next, we have to realize how a matching in G interact with an odd length cycle which is computed by our algorithm (i.e., blossom). In particular, assume that the free vertex in the cycle is unmatched.

To get a maximum number of edges of the matching in the cycle, we must at most $(n - 1)/2$ edges in the cycle, but then we can rotate the matching edges in the cycle, such that any vertex on the cycle can be free.



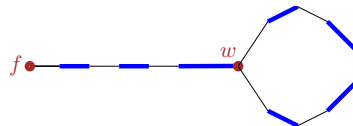
Let G/C denote the graph resulting from collapsing such an odd cycle C into single vertex. The new vertex is marked by $\{C\}$.



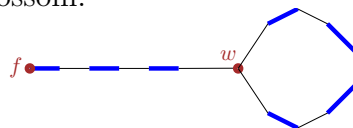
Lemma 1.3 *Given a graph G , a matching M , and a blossom B , one can find a matching M' with the same cardinality, such that B contains an alternating odd cycle with the free vertex of B being unmatched.*

Proof: If B is an alternating cycle then we are done. Otherwise, B was created by an alternating path that starts from a free vertex v , till the base vertex w of the cycle in B . Let π denote this path. Observe that the matching $M' = M \oplus \pi$ is of the same cardinality, and the cycle in B now becomes an alternating odd cycle, with a free vertex.

The blossom:



The inverted stem of the blossom:



Theorem 1.4 *Let M be a matching, and let C be an alternating odd length cycle, with the free vertex being unmatched. Then, M is a maximum matching in G if and only if M/C is a maximum matching in G/C .*

Proof: Let G/C be the collapsed graph, with $\{C\}$ denoting the vertex that correspond to the cycle C .

Note, that the collapsed vertex $\{C\}$ in G/C is free. Thus, an augmenting path π in G/C either avoids the collapsed vertex $\{C\}$ altogether, or it starts or ends there. In any case, we can rotate the matching around C such that π would be an augmenting path in G . Thus, if M/C is not a maximum matching in G/C then there exists an augmenting path in G/C , which in turn is an augmenting path in G , and as such M is not a maximum matching in G .

Similarly, if π is an augmenting path in G and it avoids C then it is also an augmenting path in G/C , and then M/C is not a maximum matching in G/C .

Otherwise, since π starts and ends in two different free vertices, and C has only one free vertex, it follows that π has an endpoint outside C . Let v be this endpoint of π and let u be the first vertex of π that belongs to C . Let σ be the path $\pi[v, u]$. ■ Let f be the free vertex of C . Note that f is unmatched. Now, if $u = f$ we are done, since then π is an augmenting path also in G/C . Note that u is matched in C , as such, it must be that the last edge e in π is unmatched. Thus, rotate the matching M around C such that u becomes free. Clearly, then σ is now an augmenting path in G and also an augmenting path in G/C .

[Note, that the rotation was only necessary as a conceptual tool, the proof works without it.]

In particular, we proved the following:

Theorem 1.5 *Let M be a matching, and let C be an alternating odd length cycle with the unmatched vertex being free. Then, there is an augmenting path in G if and only if there is an augmenting path in G/C .*

So, this suggests a very natural algorithm for computing maximum matching. Start from the empty matching M in the graph G .

Now, repeatedly, compute the graph H (this is the graph where all the free vertices are collapsed into a single vertex), and compute an alternating BFS in H . From the alternating BFS, we can extract the shortest alternating cycle based in the root. If this alternating cycle corresponds to an alternating path in G then we are done, as we can just apply this alternating path.

If this is a blossom, with a stem ρ and an odd length alternating cycle C , then apply the stem to M (i.e., compute the matching $M \oplus \rho$). Now, C is an odd cycle with the free vertex being unmatched. Compute recursively an augmenting path π in G/C . By the above discussing, we can easily transform this into an augmenting path in G . Apply this augmenting path.

Thus, we succeeded in computing a matching with one edge more in it. Repeat till the process get stuck. Clearly, what we have is a maximum size matching.

1.0.1 Running time analysis

Every shrink cost us $O(m + n)$ time. We need to perform $O(n)$ shrink till we find an augmenting path, if such a path exists. Thus, finding an augmenting path takes $O(n(m + n))$ time. Finally, we have to repeat this $O(n)$ times. Thus, overall, the running time of our algorithm is $O(n^2(m + n)) = O(n^4)$.

Theorem 1.6 *Given a graph G with n vertices and m edges, computing a maximum size matching can be done in $O(n^2m)$ time.*

2 Maximum Weight Matching in A Non-Bipartite Graph

This the hardest case, and it is non-trivial to handle. There are known polynomial time algorithms, but my feeling was that they are too involved, and somewhat cryptic, and as such should not be presented in class. For the interested student, a nice description of such an algorithm is presented in

Combinatorial Optimization - Polyhedral and efficiency by Alexander Schrijver Vol. A, 453–459.
--

The description above also follows loosely the same book.