CS 435 : Linear Optimization — Fall 2008

Lecture 18: An Algorithm for Matching

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Recap of previous lecture

To make the discussion of the previous lecture precise we make a few definitions.

Fix a matching M in a graph G.

Definition 1 *A path in G will be called alternating (w.r.t. M) if its edges alternate between those in M and those not in M.*

Definition 2 *A vertex in G will be called unmatched or free if there are no edges from M incident on it.*

Definition 3 *An augmenting path is an alternating path that starts from and ends on free (unmatched) vertices.*

Here is a formal theorem based on discussions done in the last lecture.

Theorem 1 *A matching M is maximum if and only if the resulting graph does not contain any augmenting path.*

Make sure that you can prove this theorem. One direction is easy. If there is an augmenting path then the matching is not maximum. For the other direction assume that M is not maximum. Why should an augmenting path exist? Start your proof this way: Let M_0 be an optimum matching.

Algorithm for maximum matching: We start with a single edge in the matching M. In each iteration we first find an augmenting path P. We then exchange the matched and unmatched edges in P to get a matching of one greater size. We do this till a matching with no augmenting path is found. Theorem 1 guarantees that this matching is maximum.

The crucial design step then is: How do we find an augmenting path if one exists? Note that we need to be systematic in our search otherwise we can spend a lot of time doing this.

How do we find any path, say between two vertices. One way is breadth-first-search. Can we modify this to suit our needs?

Before we answer this, we will restrict our input graphs. Recall that a graph is called bipartite if there is a partition of the vertex set into two parts such that edges are present only between the two parts. No edge has both end-points in one part. We will restrict our inputs to bipartite graphs.

For bipartite graphs an obvious modification of bfs works.

Here it is.

Modified BFS Algorithm
An augmenting path in G w.r.t. a matching M, is found using the following algorithm:

Let L be set of unmatched vertices in G w.r.t. M.
For each $u \in L$
1. Start BFS with root node u. The root node is at depth 0.
2. Till BFS is complete
   Let BFS be currently at vertex v

if v is at even depth
The children of v in the BFS tree will be unvisited vertices W such that vw is an unmatched edge in G.
else
{ v is at odd depth}
if an augmenting path exists between u and v
An augmenting path exists between u and v and can be traced.
else
The child of v in the BFS tree will be an unvisited vertex w such that vw is a matched edge.

Two questions: Why does this work for bipartite graphs? Why does this not work for non-bipartite graphs? You will find the answers to both questions in the next section.

2 Proof of correctness of modified BFS algorithm

If there exists an augmenting path P from u, then the modified BFS with u as root node must find an augmenting path for the algorithm to be correct. In fact, the given algorithm will find the shortest augmenting path P from u.
Let $\{u_0, u_1, ...u_k\}$ be the order of vertices in a shortest augmenting path P, with $u = u_0$ as one end-point.

The following lemma proves correcness of the algorithm.

LEMMA 2 *The vertex u_i will appear in the i^{th} level of the BFS tree. And vertices $u_j, j > i$ will not.*

PROOF: By induction on i.

Base Case: $i = 0$

Inductive Step: Assuming the induction hypothesis holds upto level l, consider the level $l + 1$.

Suppose l is even. Since (u_l, u_{l+1}) is a matched edge, u_{l+1} will appear in level $l + 1$. It could not have appeared earlier by the inductive assumption. Also no other vertex appears as a child of u_l.

Suppose now that l is odd. All neighbours of u_l which have not been marked yet will appear at the next level. The vertex u_{l+1} will appear at the $l + 1$th level. Also no other vertex $u_j, j > l + 1$ will appear at this level. If j were even, then the appearence of u_j in the next level will lead to a shorter augmenting path. Why? If j were odd and if u_j appears at the next level then the graph is not bipartite. Why? □

3 Time Complexity

Since the matching increases in size in every iteration, the number of iteration is bounded by $n/2$. In each iteration we may have to perform $O(n)$ searches (from unmatched vertices) before finding an augmenting path. A single run of modified BFS takes $O(m)$ time. Consider graph $G(V, E)$ and let $|V| = n$ and $|E| = m$. The time taken is at most $O(m * n^2)$. Try improving this to $O(mn)$? With some modifications a better algorithm with a run-time of $O(n^{1/2} * m)$ can be obtained.

4 Extending the algorithm for general graphs

Where does the proof of correctness fail for general graphs and why is this so crucial?

If we use the modified BFS on non-bipartite graphs, it is possible that the alogrithm might not find an augmenting path. Such an example is shown in the Fig. 1. It is tempting to modify the search slightly and hope that it works. But, as we shall see, we will need a new idea to make this work.

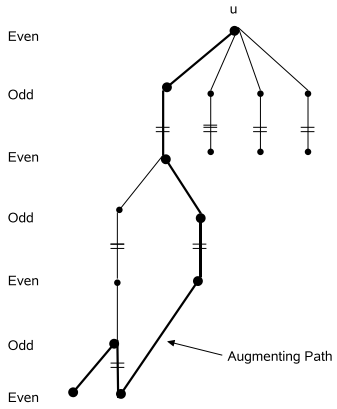

Figure 1: Modified BFS on General graph