CS 435 : Linear Optimization Fall 2008

Lecture 23: Bipartite Matching

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

A *matching* in a graph is a subset of the edges such that no two edges share an end-point. In this lecture we will design an algorithm for the maximum matching problem in bipartite graphs.
Matching
Input: A graph.
Output: A matching of maximum size.
Bipartite Matching
Input: A bipartite graph $< A, B; E >$, where A and B are the two partitions.
Output: A matching of maximum size.

We begin with a candidate ILP. One variable per edge, x_{uv}. If u, v is picked, $x_{uv} = 1$ else $x_{uv} = 0$.

The cost function is max $\sum_{u,v} x_{uv}$ The constraint is this. For each vertex v, $\sum_u x_{\{u,v\}} \leq 1$. This signifies that the number of edges incident on v is at most 1. The integrality constraint is that the variables should be either one or zero. We replace this with $x_{uv} \geq 0$, to get an LP.

Exercise. Give an example of a graph where the integral optimum is strictly less than the LP optimum for the matching problem on general (non-bipartite) graphs.

We ignore the implication of the exercise and try and design an algorithm using the method we have been following. Here we will work with the primal itself. Note that there are no weights, at least not yet!

Initialize the first matching M_0 to the emptyset. For the iterative step, we have a Matching M_i and would like to increase its size. We note that the constraint is tight for all end-points of the edges in M_i. We pick an edge $e = \{u_1, u_2\}$ with at least one end-point outside M_i. If both end-points are not matched, we may simply raise x_e to 1. Assume that u_1 is unmatched but u_2 has a matched edge (edge $\{u_2, u_3\}$ from M_i) incident on it. We wish to raise x_e by one. This implies we have to decrease x_{u_2,u_3} to zero. Then we may raise x_{u_3,u_4} for some edge incident on u_3 and so on. We see that we are exploring a path $u_1, u_2, u_3, \ldots$ such that u_i, u_{i+1} where i is even is a matched edge. This process can end in two ways. Either the last vertex is u_k with k odd and there is no matched edge incident on u_k or k is even and all edges incident on u_k are incident on some other vertex in the path. If it is the latter, we cannot really go any further. However, with the former we are able to increase the size of the matching by one. The edges in the odd places in the path enter the matching and the edges in the even places leave the matching.

Definition. Given a graph and a matching, an *alternating path* is a path with alternate edges in the matching.

Definition. Given a graph and a matching, an *augmenting path* is an alternating path such that the end-points of the path are unmatched, that is there is no edge in the matching incident on these two vertices.

The algorithm for matching, detailed above, is as follows. Start with the empty matching. For the current matching, find an augmenting path and augment the matching by adding the odd edges to the matching and removing the even edges from the matching. Continue this till such a path cannot be found.

We take up the issue of correctness first. Why is the above algorithm correct? That is, when the algorithm terminates why do we have a maximum matching? Here is a combinatorial argument. Let M be the matching found by the algorithm. Let M_o be a matching of larger size; towards a contratiction. Consider the edges in $M \oplus M_o$. These will be alternating paths of various lengths. At least one of these paths will have more M_o edges than M edges. This is an augmenting path yielding a contradiction.

For efficiency we need to design the path finding carefully. We will only get polynomial time in this lecture. We will also assume that you know how to implement breadth first search efficiently. Given a matching, and a vertex v, we need to find an augmenting path if one exists. We can then check every vertex to see if we can find an augmenting path. The key idea is modified breadth first search. We note that even edges (edges at even distance from v) are matched and odd edges are unmatched. So from v we look out to all neighbours. Say $u_1, \ldots, u_k$. From u_i we look at the matched neighbours. Say $w_1, \ldots, w_k$. So $\{u_i w_i\}$ belongs to the matching. Now from each w_i look out to all unexplored neighbours and so on. In other words, construct a breadth first search tree so that at even levels we only look at edges in the matching and at odd levels we only look at unmatched edges. One can prove by induction that if the shortest alternating path from v to u is of length k from v then the vertex u will appear in the modified bfs tree at the $k+1$th level. As in the bfs case, we can find an augmenting path in $O(m)$ time. Why? Let me point out that if you have to do a bfs from every vertex then you will only have $O(mn)$. But this can be combined. How? This gives a $O(mn)$ algorithm for bipartite matching. One can do much better by being careful. But that is another story.