CSC 373 Lecture 15

- Review: Transforming max (size) bipartite matching to max flow. A comment on the max residual capacity FF algorithm.
- Proof of the correctness of this transformation and why it does not work for weighted bipartite matching.
- Another immediate application of max flow- maximizing the number of edge (and node) disjoint paths.
- Image segmentation; binary and metric labeling

Many ways to insure polynomial time

- As claimed, one can have bad ways to choose augmenting paths such that with non rational capacities, the FF algorithm will not terminate.
- The max flow min cut theorem along with the observation that each augmenting path has residual capacity at least one insures that with integral capacities FF must always terminate. However, as the example in the notes shows there are bad ways to choose augmenting oaths so that FF does not terminate in polynomial time.

Some ways to achieve poly time

- Choose an augmenting path having shortest distance; this is the Edmonds Karp method and can be found in CLRS (page 661); it has running time *O(n m^2); n = |V|, m = |E|*
- There is a "weakly polynomial time" algorithm in Kleinberg and Tardos; here the number of arithmetic operations depends on the length of the (integral) capacities. NOTE: article claims that this method does not always terminate.
- Dinitz' method which has running time *O(m n^2)*. It is also a shortest augmenting path method based on the concept of a blocking flow .

Maximum bipartite matching

- Recall that we do not know an efficient DP or any greedy optimal algorithm for computing a max bipartite matching.
- We ***transform*** the bipartite matching problem for graph *G = (V,E)* into a flow problem:

Let *G' = (V',E')* where *V' = V + {s,t}*; and *E' = E + {(s,x)| x in X + {(y,t)| y in Y}*

We make this into a flow network *F_G* by making s and t the source and terminal nodes and setting *c(e) = 1* for all *e* in *E'*.

Transformation preserves solutions

- Every matching *M* in *G* gives rise to an integral flow *f_M* in *F_G* with *val(f_M) = |M|* and conversely every integral flow *f* in *F_G* gives rise to a matching *M_f* in *G* with |M| = val(f*)*.
- Proof: (if) *M = {<x(i1),y(j1)>,…<x(ir),y(jr)>}* is a matching of size *r*, then for *1<=k<=r*, each path *s->x(ik)->y(jk)->t* has flow 1. Since these paths are edge disjoint the total flow is *r*.
- Conversely, suppose *f* is an integral flow of value *r*. Consider an edge *<s,x(ik)>* in this flow. Since for every edge *f(e)* is integral and the capacity *c(e)* is 1, the flow into *x(ik)* is 1 and hence (by flow conservation) the flow out must be 1 and by integrality this flow must then leave on some edge *<x(ik),y(jk)>*. Now there must be *r* such *x(ik)* and the claim is that these edges *<x(ik),y(jk)>* must be a matching. Suppose there are two nodes *x(ik)* and *x'(ik)* connected to some *y(jk)*. That would give a flow of 2 into *y(jk)* but since the capacity out of *y(jk)* is 1, this would violate conservation.

Good question asked after last class

- Why can't this be modified for weighted bipartite matching? The obvious modification would set the capacity of $\langle x,y \rangle$ in E to be its weight $w(x,y)$ and the capacity of any edge $\langle e,x \rangle$ could be set to the $max\ \{w(x,y(j))\}$ and similarly for the weight of edges $\langle y,t \rangle$. Why doesn't this work? It is true that if G has a matching of total weight W then the resulting flow network has a flow of value W.
- But the converse fails! Why?

Polynomial transformation

- When we get to our next big topic (NP completeness), we will be focusing on decision problems and as a decision problem we have *|M| >=k iff val(f_M) >= k*.
- Note that we are transforming an input *(G,k)* to some *(F_G,k) = phi(G,K)* where *phi* is a function that is easily computed; i.e. certainly in time polynomial in the encoding length of the input *(G,k)*. Then we have *(G,k)* is "good" for the matching decision problem iff phi*(G,k)* is good for the flow problem.

A similar transformation

- The problem now is defined by a directed graph $G = (V,E)$ with distinguished nodes s,t. The objective is to maximize the number of edge disjoint directed paths from s to t. We basically do the same transformation, setting the capacity of all edges in E to be 1. Once again because of integrality we can argue that G has k disjoint paths iff only the transformed input F_G has (integral) flow k.
- We can adapt this transformation to work for node disjoint paths and we can also have the same result for undirected graphs.

The labeling problem

In section 12.6, the text defines the labeling problem and then provides a local search approximation algorithm (with approximation ratio ~ 2). For context I will state the problem formulation as in section 12.6 and then discuss the special case of 2 labels that is the subject of section 7.10.

The labeling problem

- The input is a graph *G = (V,E)*, a set of labels *L = {a_1, …a_r}* and two cost functions *p : E :* -> non negative reals and *c : V x L :* -> non negative reals
- c_i(a) is the cost of giving the label `*a*' to node *i* (e.g. a document)
- *p_ij* is the cost of *not* giving nodes *i* and *j* the same label. (More generally, there is a metric *d* on the labels *L* and the cost depends on how far apart are the labels given to nodes *i* and *j*. The text only considers the {0,1} metric.)

Restricting to two labels

- When there are more than 2 labels, the problem becomes NP-hard. And section 12.6 provides an approximation algorithm for this problem. But when there are only 2 labels, the problem can be solved by calculating an appropriate min cut (and hence by max flow).
- For two labels, the problem is restated as minimizing
  q'(A,B) =
  sum_{i in A} b_i + sum_{j in B} a_j +
  sum_{(i,j):A* contains exactly one of the nodes *i,j} p_ij

The transformation

- The intuitive idea is that we are trying to partition the nodes into those that will be labeled as belonging to *A* and those that belong to *B*. This is the idea of a cut in the graph and the edge costs *p_ij* across that cut will reflect the costs of giving i and j different labels. But how to reflect the costs the labels given to nodes *i* and *j*? We will do this by introducing a source s and terminal *t* and connecting *s* to every node *j* in *V* with cost *a_j* and connecting every node *i* in V with cost *b_i* and then after any assignment we have a flow network as in KT Fig 7-19