CSC 373 Lecture 12

- One last DP algorithm: Sequence alignment/ edit distance
- The difference between sequence alignment and bipartite matching
- Start Flow networks

Sequence alignment/Edit distance

- In the edit distance problem we are given two strings $X = x(1)x(2)....x(m)$ and $Y = y(1)...y(n)$ over some finite alphabet S. We are trying to find the best way to "match" these two strings. Variants of this problem occur often in bio-informatics.
- Sometimes this is cast as a maximization problem; we will view it as a minimization problem by defining different distance measures and matching symbols so as to minimize this distance.

A simple distance measure

- Suppose we can delete symbols and match symbols. We can have a cost *d(a)* to delete a symbol *a* in *S*, and a cost *m(a,b)* to match symbol *a* with symbol *b* (where we would normally assume *m(a,a) = 0*).
- As in any DP we consider an optimal solution and let consider whether or not we will match the rightmost symbols of *X* and *Y* or delete a symbol.

The DP arrays

- For the semantic array we have *E[i,j]* is the cost of an optimal match of *x(1)…x(i)* and *y(1)…y(j)*.
- *E'[0,0] = 0* ;
  E'[0,j] = E'[0,j-1] + d(y(j)) for *j > 0*;
  E'[i,0] = E'[i-1,0] + d(x(i)) for *i >0*;
  E'[I,j] = min{A,B,C} where *A = E'[i-1,j-1] + m(x(i),y(j))*,
  B = E'[i-1,j] + d(x(i)), and *C = E'[i,j-1] + d(y(j))*.
- As a simple variation of edit distance we consider the maximization problem where each "match" of "compatible" *a* and *b*) has profit 1 (resp. *v(a,b)*) and all deletions and mismatches have 0 profit. This is a special case of unweighted (resp. weighted) bipartite graph matching <u>where edges cannot cross</u>.

Flow networks

- I will be following our old CSC364 lecture notes for the basic definitions and results concerning the computation of max flows. In doing so we follows the convention of allowing negative flows. While intuitively this may not seem so natural, it does simply the development.
- A flow network (more suggestive to say a capacity network) is as follows: $F = (G,s,t,c)$ where $G = (V,E)$ is a "bidirectional graph", s (the source) and t (the terminal) are nodes in V, and c is a non negative real valued function on the edges.

A valid flow

- A flow f is a real valued function on the edges satisfying the following properties:
  1) $f(e) <= c(e)$ for all edges
  (capacity constraint)
  2) $f(u,v) = - f(v,u)$ (skew symmetry)
  3) for all nodes *u* (except for *s* and *t*), the sum of flows into (or out of) *u* is zero. (Flow conservation). Note: this is the "flow in = flow out" constraint for the convention of only having non negative flows.

The max flow problem

- The goal of the max flow problem is to find a valid flow that maximizes the flow out of the source node *s*. As we will see this is also equivalent to maximizing the flow in to the terminal node *t*. (This should not be surprising as flow conservation dictates that no flow is being stored in the other nodes.) We let $val(f) = |f|$ denote the flow out of the source *s* for a given flow *f*.
- We will study the Ford Fulkerson augmenting path scheme for computing an optimal flow. I am calling it a "scheme" as there are many ways to instantiate this scheme although I don't view it as a general "paradigm" in the way I view (say) greedy and DP algorithms.

So why study Ford Fulkerson?

- The question is why study the Ford Fulkerson scheme if it is not a very generic algorithmic approach?
- I view Ford Fulkerson and augmenting paths as an important example of a local search algorithm although unlike most local search algorithms we obtain an optimal solution.
- The topic of max flow (and various generalizations) is important because of its immediate application and because of the many applications of max flow type problems to other problems (e.g. max bipartite matching). That is, in the terminology that we will soon be using, many problems can be polynomial time transformed/reduced to max flow (or one of its generalizations). One might refer to all these applications as "flow based methods".

A flow f and its residual graph

- Given any flow *f* for a flow network *F = (G,s,t,c)*, we can define the residual graph *G(f) = (V,E(f))* where *E(f)* is the set if all edges having positive residual capacity ; that is, *c(e)-f(e) > 0*. Note that *c(e)-f(e) >= 0* for all edges by the capacity constraint. Note that with our convention of negative flows, even a zero capacity edge (in *G*) can have residual capacity.
- The basic concept underlying Ford Fulkerson is that of an augmenting path which is an *s-t* path in *G(f)*. Such a path can be used to augment the current flow *f* to derive a better flow *f'*.