CSC373: Lecture 11

Comments on term test 1?

The all pairs least cost problem and alternative DPs

The matrix chain problem

The all pairs least cost problem

- We now wish to compute the least cost path for all pairs *<u,v>* in an edge weighted directed graph (with no negative cycles).
- We can repeat the single source DP for each possible source node: complexity *O(n^4)*
- We can reduce the complexity to *O(n^3 log n)* using the DP based on the semantic array *E[j,u,v]* = cost of shortest path of path length at most *2^j* from *u* to *v*.   What is corresponding computational array?

Another DP for all pairs

- Let us assume WLG that *V = {1,2,…,n}*.
- We now define the semantic array *G[k,u,v]* to be the least cost of a (simple) path *pi* from *u* to *v* such that the internal nodes in the path pi are in the subset *{1,2,…,k}.*
- The computational array is *G'[0,u,v] = 0* if *u = v* and = *c(u,v)* if *<u,v>* is an edge; infinite otherwise. *G'[k+1,u,v]* = *min{A,B}* where *A = G'[k,u,v]* and
  B = G'[k,u,k+1] + G'[k,k+1,v]
- Like the recursion for the previous array *E'[j,u,v]*, the recursion here uses two recursive calls for each entry.
- Complexity: *n^3* entries and only *O(1)* per entry.

Two similar DPs (using 2 recursive calls); DP over intervals

- The matrix chain problem : We are given n matrices (say over some field) *M(1),…,M(n)* with *M(i)* having dimension *d(i-1) x d(i)*. The goal is to compute the matrix product *M(1)*M(2)* …*M(n)* using a given subroutine for computing a single matrix product *A*B*.
- We recall that matrix multiplication is associative; that is, *(A*B)*C = A*(B*C)* but the number of operations for computing *A*B*C* generally depends on the order in which the pairwise multiplications are carried out.

Matrix chain product continued

- Let us assume that we are using classical matrix multiplication and say that the scalar complexity for a (*p* x *q*) times (q x r) matrix multilication is pqr.
- For example say the dimensions of A,B,C are (respectively)   5x10, 10x100, 100x50. Then using (A*B)*C costs 5000 + 25000 = 30000 scalar ops whereas A*(B*C) costs 50000 + 2500 = 52500 scalar ops.
- NOTE: For this problem the input is these dimensions and not the actual matrix entries.

Parse tree for the product chain

- The matrix product problem then is to determine the parse tree that describes the order of pairwise products. At the leaves of this parse tree are the individual matrices and each internal node represents a pairwise matrix multiplication.
- Once we think of this parse tree, the DP is reasonably suggestive. That is, the root of the optimal tree is the last pairwise multiplication and the subtrees are subproblems that must must be computed optimally.

The DP arrays

- The semantic array we are led to is *C[i,j]* is the cost of an optimal parse of *M(i) * … * M(j)* for *1 <= i <= j <=* n
- The recursive computationally array is as follows: *C'[i,i] = 0* for all *i* and for *i<j,* *C'[i,j] = min{ C[i,k] + C'[k+1,j] + d(i-1)d(k)d(j): i <=k <j}*
- *This same style DP algorithm (called DP over intervals) is also used in the RNA folding problem in the text as well as in computing optimal binary search trees. Essentially in all these cases we are computing an optimal parse tree.*

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] = d(y(j)) for *j > 0*; *E'[i,0] = d(x(i))* for *i >0*;
  E'[I,j] = min{A,B,C} where *A = m(x(i),y(j)),*
  B = d(x(i)), and *C = 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>.

DP-concluding remarks

- In DP algorithms one usually has to first generalize the problem (as we did to more or less some extent for all problems considered). Sometimes this generalization is not at all obvious.
- What is the difference between divide and conquer and DP? In divide and conquer the recursion tree never encounters a subproblem more than once. In DP, we need memoization (or an iterative implementation) as a given subproblem can be encountered many times leading to exponential time complexity if done without memoization.