CSC373: Lecture 9

Questions about problem set?

Test format

One more charging argument?

A second pseudo polynomial time algorithm for the knapsack problem

Turning the pseudo polynomial time algorithm into a fully polynomial time approximation scheme (FPTAS)

DP for least cost paths problem

The Knapsack problem

- In the knapsack problem we are given a set of *n* items *I(1), …, I(n*) and a size bound *B* where where each item *I(j) = (s(j), v(j))* with *s(j)* being the size of the item and v(j) the value. (Often one uses *w(j)* for the weight of the item rather than *s(j)* but I am avoiding that due to our earlier use of w(j) which corresponded to the weight or profit of an interval in the WISP.) In general we can allow real valued parameters but in some cases need to restrict attention to integral parameters.
- A feasible set is now a subset of items *S*: the sum of the sizes of items in *S* is at most the bound *B*. The goal is to find a feasible set *S* maximizing the sum of the values of items in *S*. This is know to be an NP hard problem but as we shall see it is only "weakly" NP hard. (It remains an NP hard problem even when *v(j) = s(j)* for all *j*.)

The first DP algorithm

- The previous approach did not work because it allows using an item more than once.
- Instead we can use *V[i, b*] = the maximum profit possible using only the first *i* items and not exceeding the bound *b*.
- The corresponding computational array is : *V'[0,b] = V'[i,0] = 0. V'[i,b] = max {C,D}* where *C = V'[i-1, b]* and *D = V'[i-1, b-s(i)] + v(i)* if *s(i) <= b*; else 0
- This algorithm has running time O(nB) and is pseudo polynomial time.

A second DP knapsack algorithm

- In the first algorithm, if the weights (or the bound *B*) are small (i.e. *B = poly(n)*) then the algorithm runs in polynomial time.
- What if the values {v(i)} are integral and small? Consider the following semantic array *W[i,v]* = minimum size required to obtain profit *v* using a subset of the items *I(1), …,I(i)* if possible; infinity otherwise
- The desired optimum value is *max {v: W[n,v]}* is at most *B*.

Corresponding computational array

- *W'[0, v]* = infinite for all *v > 0*
- *W'[i,v] = 0* for all *i* and all *v <= 0*
- *W'[i,v] = min{C,D}* where *C = W'[i-1,v]* and *D= W'[i-1, v-v(i)] + s(i)*
- This DP remain pseudo polynomial time but now the complexity is *O(nV)* where *V = v(1) + v(2) …+ v(n)*.

An FPTAS for the knapsack problem

- This algorithm can be used as the basis for an efficient approximation algorithm for all input instances. The basic idea is relatively simple:
- The high order bits/digits of the values can determine an approximate solution (disregarding low order bits).
- The fewer high order bits we use, the faster the algorithm but the worse the approximation. The goal is to scale the values in terms of a parameter epsilon so that a (*1+epsilon*) approximation is obtained with time complexity polynomial in *n* and (*1/epsilon*). The details are given in the KT text (section 11.8).

Looking ahead toward discussion of NP complete problems

- In term of computing optimal solutions, all ``NP complete optimization problems'' (i.e. optimization problems corresponding to NP complete decision problems) can be viewed (up to polynomial time) as a single class of problems.
- But in the world of approximation algorithms, this single class splits into many classes of approximation guarantees. Up to our believed complexity assumptions these possibilities include:

Different approximation possibilities for NP complete optimization given widely believed complexity claims

- An FPTAS (e.g. knapsack problem)
- A PTAS but no FPTAS (makespan)
- Having a $c > 1$ approximation but no PTAS (JISP)
- An O($\log n$) approximation but no constant approximation (set cover)
- No $n^{1-\epsilon}$ approximation for any $\epsilon > 0$. (graph colouring and MIS for arbitrary graphs)
- Here n stands for some input size parameter (e.g. number of nodes in a graph)

A DP with a somewhat different style

- Lets consider the single source least cost paths problem which is efficiently solved by Dijkstra's greedy algorithm for graphs in which all edge costs are non-negative.
- The least cost paths problem is still well defined as long as there are no negative cycles; that is, the least cost path is a simple path.

Single source least cost paths for graphs with no negative cycles

- Following the DP paradigm , we consider the nature of an optimal solution and how it is composed of optimal solutions to ``subproblems''.
- Consider an optimal simple path *P* from source *s* to some node *v*. This path could be just an edge but if the path *P* has length greater than 1, then there is some node *u* which immediately proceeds *v* in *P*. If *P* is an optimal path to *v*, then the path leading to *u* must also be an optimal path.
- We are led to define the following semantic array: *C[i,v]* = the minimum cost of a simple path with path length at most *i* from source *s* to *v*. (If there is no such path then this cost is infinte.)
- The desired answer is then the single dimensional array derived by setting $i = n-1$ *where* $n = |V|$.

Corresponding computational array

- *C'[0,v] = 0* if *v = s* and infinite otherwise.
- *C'[i,v] = min {A,B}* where *A = C'[i-1,v]* and *B = min {C'(i-1, u) + c(u,v) | (u,v) in E}* .
- Note: This presentation is slightly different than in the KT text.
- Why is this a slightly different form than before? Namely, showing the equivalence between the semantic and computationally defined arrays) is not an induction on the number of input items in the solution but is based on some other parameter (i.e. the path length) of the solution.