CSC 373: Algorithm Design and Analysis
Lecture 7

Allan Borodin

January 21, 2013

Lecture 7: Outline

- A second pseudo polynomial time algorithm for the knapsack problem
- Turning a pseudo polynomial time algorithm into a fully polynomial time approximation scheme (FPTAS)
- DP for the least cost paths problem

Review of DP for knapsack problem from last lecture

The Knapsack problem

- In the knapsack problem we are given a set of n items $I_1, \ldots, I_n$ 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.
- A feasible set is now a subset of items S such that the sum of the sizes of items in S is at most the bound B.
- Goal: Find a feasible set S maximizing the sum of the values of items in S.

- 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 to denote the weight or profit of an interval in the WISP.
- In general we can allow real valued parameters but in some algorithms need to restrict attention to integral parameters. But by scaling inputs this is not a significant restriction.
- This is known to be an NP hard problem but as we shall see it is only "weakly NP hard". However, It remains an NP hard problem even when $v_j = s_j$ for all j.

The first DP algorithm for the knapsack problem

- Define

$V[i, b]$ = the maximum profit possible using only the first i items and not exceeding the bound b.

- The corresponding computational array is :

$$V'[i, b] = \begin{cases} 0 & \text{if } i = 0 \text{ or } b = 0 \\ \max\{C, D\} & \text{if } s_i \leq b \end{cases}$$

where

$$C = V'[i-1, b] \text{ and } D = V'[i-1, b-s_i] + v_i.$$

- This algorithm has running time $O(nB)$ and is pseudo polynomial time.

- Question: why is it not polynomial time?

A second DP algorithm for the knapsack problem

- In the first algorithm, if the sizes (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

$$W[i, v] = \begin{cases} \text{minimum size required to obtain at least profit } v \text{ using} \\ \qquad\qquad \text{a subset of the items } \{I_1, \ldots, I_i\} \text{ if possible} \\ \infty \text{ otherwise} \end{cases}$$

- The desired optimum value is $\max\{v : W[n, v] \text{ is at most } B\}$.

Corresponding computational array

- The corresponding computational array is :

$$W[i,v] = \begin{cases} \infty & \text{if } i = 0 \text{ and } v > 0 \\ 0 & \text{if } i \leq 0 \text{ and } v \leq 0 \\ \min\{C, D\} & \text{otherwise.} \end{cases}$$

  where

$$C = W[i-1, v] \text{ and } D = W[i-1, v-v_i] + s_i.$$

- This DP remains pseudo polynomial time but now the complexity is $O(nV)$ where $V = v_1 + v_2 + \ldots + v_n$.

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 after rounding up).
  - 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 ϵ so that a $(1+\epsilon)$ approximation is obtained with time complexity polynomial in n and $(1/\epsilon)$.
  - The details are given in the DPV text (section 9.2.4) or the KT text (section 11.8).
  - Namely, KT set $\hat{v}_i = \lceil \frac{v_i n}{\epsilon v_{max}} \rceil$ where $v_{\max} = \max_j\{v_j\}$. DPV use the floor $\lfloor\ \rfloor$.
  - The running time is $O(n^3/\epsilon)$.

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, we next discuss these possibilities.

Definition

1. An FPTAS (Fully Polynomial Time Approximation Scheme) algorithm is one that is polynomial time in the encoding of the input and $\frac{1}{\epsilon}$.
2. A PTAS (Polynomial Time Approximation Scheme) algorithm is one that that is polynomial in the encoding of the algorithm but can have any complexity in terms of $\frac{1}{\epsilon}$.

Different approximation possibilities for NP complete optimization

Given widely believed complexity claims

1. An FPTAS
   - e.g. the knapsack problem
2. A PTAS but no FPTAS
   - e.g. makespan (when the number of machines m is not fixed but rather is a a parameter of the problem.
3. Having a constant $c > 1$ approximation but no PTAS
   - e.g. JISP
4. An $\Theta(\log n)$ approximation and no constant approximation
   - e.g. set cover H_n essentially tight.
5. No $n^{1-\epsilon}$ approximation for any $\epsilon > 0$
   - e.g. graph colouring and MIS for arbitrary graphs

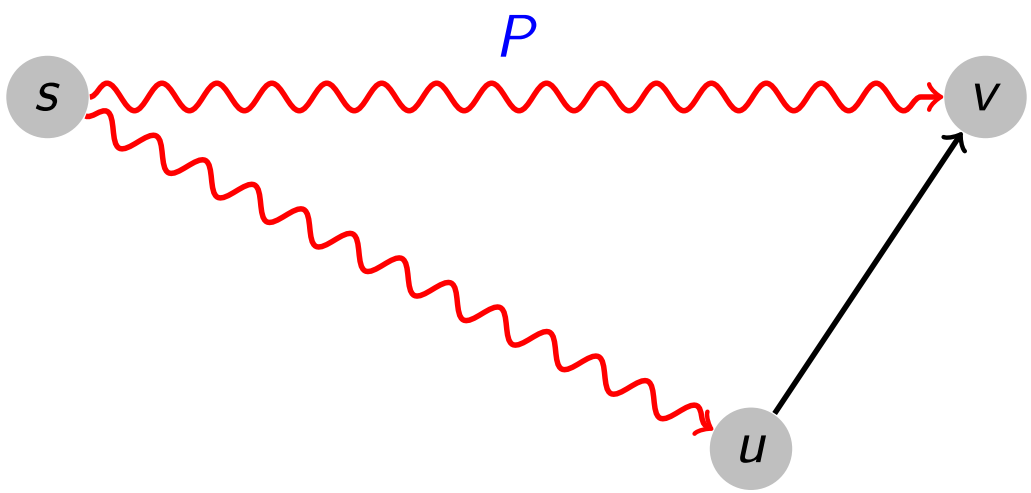
Here n stands for some input size parameter (e.g. size of the universe for set cover and number of nodes in the graph for colouring and MIS).

A DP with a sightly different style

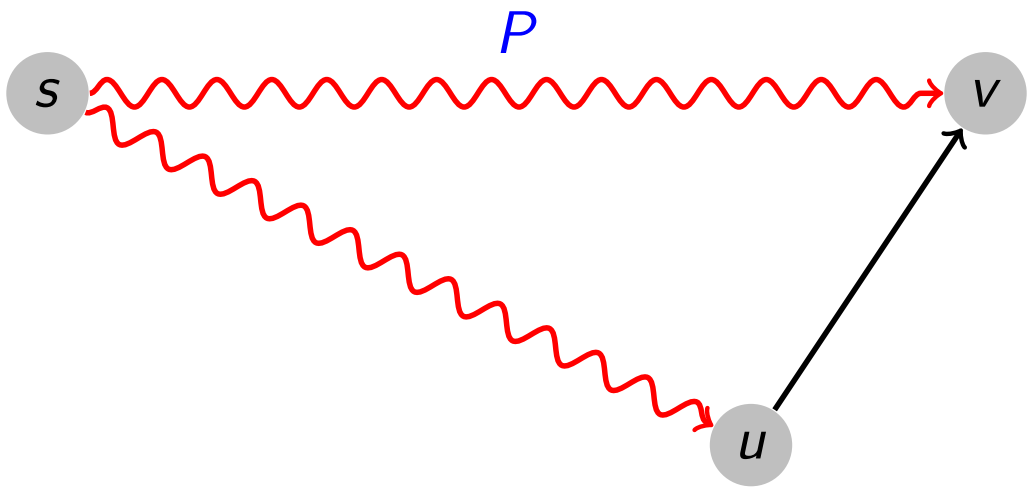
- Let's consider the single source least cost paths problem which is efficiently solved by Dijkstra's greedy algorithm for graphs in which all edge costs are non-negative.
- The least cost paths problem is still well defined as long as there are no negative cycles; that is, the least cost path is a simple path.
- The text presents the Bellman-Ford algorithm in Chapter 4 but it can be (and I think is best) presented as a DP algorithm and we will present it within this context.
- The algorithm can also be thought of as an adaptive greedy algorithm if we consider the edges as the input items. But still I think the DP point of view is what leads us to this algorithm.

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.



Single source least cost paths for graphs with no negative cycles



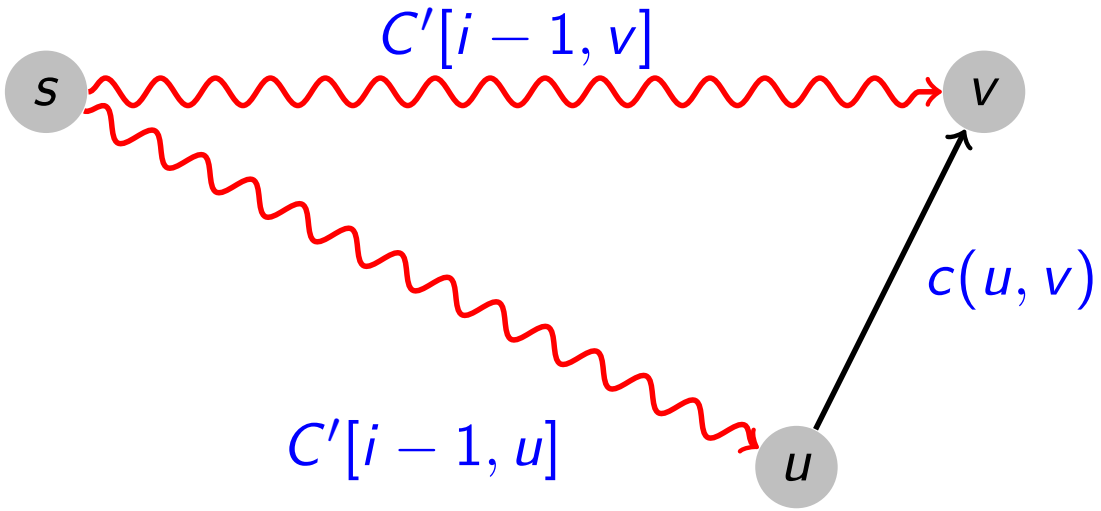
- This leads to the following semantic array:

$C[i, v]$ = 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 ∞.)

- The desired answer is then the single dimensional array derived by setting $i = n - 1$. (Any simple path has path length at most $n - 1$.)

How to construct the computational array?

- We can construct $C'[i, v]$ from $C'[i-1, \ldots]$ as follows:



- Let $C'[i, v]$ be the minimum value among
  - $C'[i-1, v]$
  - $C'[i-1, u] + c(u, v)$ for all $(u, v) \in E$.

Corresponding computational array

- The computational array is defined as:

$$C'[i,v] = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = s \\ \infty & \text{if } i = 0 \text{ and } v \neq s \\ \min\{A, B\} & \text{otherwise} \end{cases}$$

$$A = C'[i-1, v]$$

$$B = \min\left\{ C'[i-1, u] + c(u,v) : (u,v) \in E \right\}$$

- Why is this slightly different from before?
  - Namely, showing the equivalence between the semantic and computationally defined arrays is not an induction on the indices of the input items in the solution.
  - But it is based on some other parameter (i.e. the path length) of the solution.
- Time complexity: n^2 entries $\times$ $O(n)$ per entry $= O(n^3)$ in total.