Solutions to Problem Set 7

Rashmi Dwaraka

December 10, 2016

Collaborator: Nakul Camasumudram, Shabbir Hussain

Contents

1 Complexity of clustering

Problem statement:
Clustering of similar documents is an important technique often used in information retrieval applications. It is common to define similarity by means of a distance function. Let S denote a set of documents. For any two documents X and Y , let $d(X, Y)$ denote the distance between them. (We will assume that $d(X, Y)$ is a nonnegative integer.) The smaller the distance, the more similar the two documents are.

Consider the LargestCluster problem: given a set S of documents, a distance function d, and an integer r, determine the largest subset T of S such that $d(X, Y) \leq r$ for any two documents X and Y in T.

Formulate a decision version of LargestCluster and prove that it is NP-complete. (Hint: You may use a reduction from the decision version of the Clique problem.)

Solution:

Decision problem:
Input:

- Set S of documents
- For any two documents X and Y , let d(X, Y) denote the distance between them
- An integer r
- LargestCluster T of size K

Output:

- Yes, if $\exists$ T $\subseteq$ S such that
  - $\forall$ X,Y $\in$ T : d(X,Y) $\leq$ r
  - |T| $\leq$ k
- No, otherwise

Claim: LargestCluster Problem $\in$ NP-Complete

- **Certificate:** LargestCluster T
- **Verifier:** For all pairs in T, check if distance between them $\leq$ r
  The certificate can be veriified in $O(n^2)$ polynomial time, where $n = |T|$
- **Reduction:** Clique $\leq$p LargestCluster Decision
  Given an instance of graph G $= < V, K > \rightarrow < V, d(X, Y) = 1$, if X,Y is an edge
  $d(X, Y) = 2$, if X,Y is not an edge, r=1,$K = K >$
  Clique C in G $\iff$ Cluster with pair-wise distance 1

2 Longest simple paths

Problem statement:
You are given a directed graph $G = (V, E)$ with integer weights on edges.

(a) Suppose G is a directed acyclic graph. Give a polynomial-time algorithm to determine a simple path in G that has largest weight.

(b) Suppose G is an arbitrary directed graph. In the longest simple path problem, you are also given an integer L and asked if there exists a simple path in G that has weight $\geq L$. Use a reduction from the Directed Hamiltonian Path Problem to show that the longest simple path problem is NP-complete.

Solution:

(a) In a DAG, all path are simple path. Add a pseudo Source vertex that connects to all vertices of the DAG. A longest path through the DAG can be found either

- by negating the edge weights and running DAG-SHORTEST-PATHS, or
- running DAG-SHORTEST-PATHS, with the modification that we replace ∞ by -∞ in line 2 of INITIALIZE-SINGLE-SOURCE and $>$ by $<$ in the RELAX procedure.

Based on the d and π attribute of the vertices set in the above algorithm, we can backtrack and trace the longest simple path in the DAG.
The running time of the algorithm is $\Theta(|V|+|E|)$
Reference: CLRS 24.2.

(b) The Longest Path Problem is in NP.
Cetificate: A solution path P
Verifier: Check that P consists of at least k edges, and that these edges form a simple path (no vertex is used more than once). This verification can be done in polynomial time.
Reduction: The reduction follows directly from Hamiltonian Path. Given an instance of Hamiltonian Path on a graph G = (V, E). We create an instance of the longest path problem $< G_0, k >$ where all the edge weights are set to 1.
Using the graph G_0, set k = $|V|$ - 1. Then there exists a simple path of length k in $G_0 \iff$ G_0 contains a Hamiltonian path.

3 Scheduling with profits and deadlines

Problem statement:
Suppose that we have one machine and a set of n tasks $a_1, a_2, ..., a_n$, each of which requires time on the machine. Each task a_j requires t_j time units on the machine (its processing time), yields a profit of p_j, and has a deadline d_j . The machine can process only one task at a time, and task a_j must run without interruption for t_j consecutive time units. If we complete task a_j by its deadline d_j,we receive a profit p_j , but if we complete it after its deadline, we receive no profit. As an optimization problem, we are given the processing times, profits, and deadlines for a set of n tasks, and we wish to find a schedule that completes all the tasks and returns the greatest amount of profit. The processing times, profits, and deadlines are all non-negative numbers.

a. State this problem as a decision problem.

b. Show that the decision problem is NP-complete.

c. Give a polynomial-time algorithm for the decision problem, assuming that all processing times are integers from 1 to n. (Hint: Use dynamic programming.)

d. Give a polynomial-time algorithm for the optimization problem, assuming that all processing times are integers from 1 to n.

Solution:

a. The Decision problem of scheduling the tasks can be as follows:
Does there exist a permutation s Sn such that if we run the tasks in the order s[1],...,s[n], the total profit is at least k?

b. The Decision problem is NP-C and proof is as below:

Certificate: The permutation of tasks S[1..n].
Verifier: Given an instance of x[1..n] and P, we can check for any overlapping tasks and if the total profit obtained is $\geq$ K.
The verifier can be verified in polynomial time.

Reduction:
Subset Sum problem can be reduced to this scheduling problem and prove that the scheduling problem is NP.
Given an instance of the SubsetSum problem A = (x[1...n], t), it can be formulated into an instance B in polynomial time as below:

- T[1...n] = P[1...n] = x[1...n]
- D[1...n] = t
- K = t

If B defined above has a schedule for the subset S of tasks whose completion time is within the deadline:

- $\sum_{i \in S} x_i \leq t$
- $\sum_{i \in S} x_i \geq t$
- which implies $\sum_{i \in S} x_i = t$

From the above we can say that there is a subset of elements in A, such that their sum is exactly t. Suppose if there is a subset of elements in A whose sum is exactly equal to t, then there is a corresponding schedule in B.

c. Using the dynamic programming solution suggested below, we can deduce that if the optimal profit $P_{opt} \geq K$, return yes, Else return no.

d. The dynamic programming solution can be used with the following recurrence:
Let maxP(t,i) is the maximum profit obtained by scheduling jobs i...n, where the scheduling starts at time t.

$$maxP(t,i) = max \begin{cases} 0, & \textbf{if i>n} \\ P[i] + maxP(t+T[i], i+1), & \textbf{if } (t+T[i]) \leq \textbf{D[i] and i<n} \\ maxP(t, i+1), & otherwise \end{cases}$$

Algorithm:
Let the arrays T, D, P be defined Globally and D $\leftarrow$ sorted in increasing order of deadlines in D. The solution would be maxP(0,1)

Algorithm: maxP(t,i)
Global:Profit[][] matrix of size $D_{max} \times n$ **for memoization**

1: **procedure: maxP**(t,i)
2: if Profit$[t][i] \neq$ NULL
3: return Profit$[t][i]$
4: else if $(i > n)$
5: return 0
6: else if $((t + T[i]) \leq D[i])$
7: return $max(maxP(t + T[i], i + 1) + P[i], maxP(t, i + 1)))$
8: else return maxP$(t, i + 1)$

Worst case Running time:
The runtime of the algorithm depends on the size of the Profit matrix used for memoization. The Processing time for each of the n jobs is an integer from 1 to n and hence the maximum time to complete all the jobs is atmost n^2 which implies that D_{max} is n^2. The maximum possible deadline D_{max} is $O(n^2)$, and the size of the matrix is $D_{max} \times n$. Therefore the total runtime is $O(n^3)$.