CSE 202: Design and Analysis of Algorithms **(Due: 12/06/19)**

Homework #4

Instructor: Ramamohan Paturi *Name:* Shihan Ran, *Netid:* A53313589

Problem 1: Hamiltonian path

Problem Description

Suppose we are given a directed graph $G = (V, E)$, with $V = \{v_1, v_2, \ldots, v_n\}$, and we want to decide whether G has a Hamiltonian path from v_1 to v_n. (That is, is there a path in G that goes from v_1 to v_n, passing through every other vertex exactly once?)

Since the Hamiltonian Path Problem is NP-complete, we do not expect that there is a polynomial-time solution for this problem. However, this does not mean that all nonpolynomial-time algorithms are equally "bad." For example, here's the simplest brute-force approach: For each permutation of the vertices, see if it forms a Hamiltonian path from v_1 to v_n. This takes time roughly proportional to $n!$, which is about 3×10^{17} when $n = 20$.

Show that the Hamiltonian Path Problem can in fact be solved in time $O(2^n \cdot p(n))$, where $p(n)$ is a polynomial function of n. This is a much better algorithm for moderate values of n; for example, 2^n is only about a million when $n = 20$.

In addition, show that the Hamiltonian Path problem can be solved in time $O(2^n \cdot p(n))$ and in polynomial space.

Solution

(High-level description)

Given $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, we want to decide if there is a Hamiltonian path from v_1 to v_n. We regard $H(G, v_1, v_n)$ as the solution of the question. Clearly, $H(G, v_1, v_n)$ is true if and only if there is a vertex v_k with $H(G - \{v_1\}, v_k, v_n)$ is true, and there exist an edge (v_1, v_k) so that we can get $H(G, v_1, v_n) = 1$ by simply add the path (v_1, v_k) to path (v_k, v_n).

Following the idea, we can construct the answer starting from the largest sets and gradually working down to smaller ones. It's kind of like dynamic programming.

The recursive formulation should be:

$$H(G, v_i, v_j) = \max_k H(G - \{v_i\}, v_k, v_j) \text{ and } (v_i, v_k) \in E$$

(Correctness)

Base Case: The base case is the graph G only contain 2 vertices v_1 and v_2, at least one edge exists between v_1 and v_2. Hence, $H(G, v_1, v_2) = 1$.

The recursive formulation is correct because of that any Hamiltonian path from v_i to v_j can be divided into one Hamiltonian path v_k to v_j plus one edge (v_i, v_k). Hence the recursive formulation is true. By considering all the potential vertices, and following the dynamic programming, we ensure the H is correctly computed.

Correctness proved.

(Time complexity)

We need $O(2^n)$ to decide the subset graph. And determining if $H(G - \{v_i\}, v_k, v_j) = 1$ takes $O(n)$. Thus, the total time complexity should be $O(2^n \cdot n^2) = O(2^n \cdot p(n))$.

Since we follows the idea of dynamic programming, we store each time step's calculated results $H(G - \{v_i\}, v_k, v_j)$. Hence, after chosen v_1 and v_n, the space complexity should be $O(n)$. The total space complexity is $O(2^n \cdot n)$.

Problem 2: Remote Sensors

Problem Description

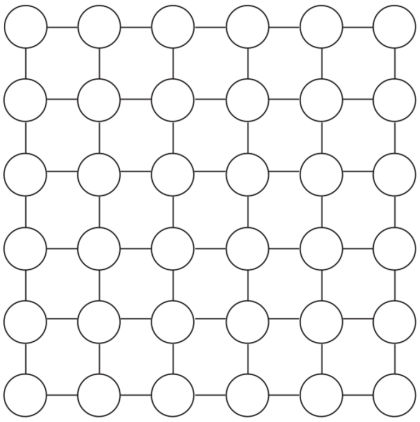

Figure 1: A grid graph

Suppose you are given an $n \times n$ grid graph G, as in Figure 1 Associated with each node v is a weight $w(v)$, which is a nonnegative integer. You may assume that the weights of all nodes are distinct. Your goal is to choose an independent set S of nodes of the grid, so that the sum of the weights of the nodes in S is as large as possible. (The sum of the weights of the nodes in S will be called its total weight.) Consider the following greedy algorithm for this problem.

Algorithm 1: The "heaviest-first" greedy algorithm

Start with S equal to the empty set
while some node remains in G **do**
  Pick a node v_i of maximum weight
  add v_i to S
  Delete v_i and its neighbors from G
end while
return S

(Subproblem 1)

Let S be the independent set returned by the "heaviest-first" greedy algorithm, and let T be any other independent set in G. Show that, for each node $v \in T$, either $v \in S$,or there is a node $v' \in S$ so that $w(v) \leq w(v')$ and (v, v') is an edge of G.

(Solution 1)

Proof. For node $v \in T$, if $v \notin S$, then it means that it is deleted at some iteration when selecting its neighbor v'. And following the instruction of greedy algorithm, v' must contain a larger weight than v. Otherwise we will choose to add v to S. Hence, we proved that for each node $v \in T$, either $v \in S$ or there is a node $v' \in S$ so that $w(v) \leq w(v')$ and (v, v') is an edge of G.

(Subproblem 2)

Show that the "heaviest-first" greedy algorithm returns an independent set of total weight at least 1/4 times the maximum total weight of any independent set in the grid graph G.

(Solution 2)

We use T to denote any independent set in the grid graph G, let S denote the independent set found by the heaviest-first greedy algorithm. We want to show that $W_S \geq \forall_T \frac{1}{4} W_T$.

For any node $v \in T$,

Case 1. If $v \in S$, since G is a grid graph, then at most four neighboring nodes of no greater weights $v'_{1,2,3,4} \in T$.

Case 2. If $v \notin S$, as proved in subproblem 1, there must be a node $v' \in S$ so that $w(v) \leq w(v')$ and (v, v') is an edge of G. For v', there are at most four neighboring nodes of no greater weights $\in T$.

Either case, the total weight of all nodes in T should not be greater than four times than the total weights of all nodes in S. Hence, we proved that $W_S \geq \forall_T \frac{1}{4} W_T$.

Problem 3: Scheduling

Problem Description

Consider the following scheduling problem. You are given a set of n jobs, each of which has a time requirement t_i. Each job can be done on one of two identical machines. The objective is to minimize the total time to complete all jobs, i.e., the maximum over the two machines of the total time of all jobs scheduled on the machine. A greedy heuristic would be to go through the jobs and schedule each on the machine with the least total work so far.

(Subproblem 1)

Give an example (with the items sorted in decreasing order) where this heuristic is not optimal.

(Solution 1)

5 jobs with time $10, 9, 8, 7, 4$.

The optimal method should be assigning machine one with $10, 9$ and machine two with $8, 7, 4$. The optimal total time is 19.

However, following the greedy heuristic, the scheduling should be assigning machine one with $10, 8, 4$, machine two with $9, 7$. The time should be 22. Hence, this heuristic is not optimal under this example.

(Subproblem 2)

Assume the jobs are sorted in decreasing order of time required. Show as tight a bound as possible on the approximation ratio for the greedy heuristic. A ratio of 7/6 or better would get full credit. A ratio worse than 7/6 might get partial credit.

(Solution 2)

We regard OPT as the optimal solution of minimum total time to complete all jobs, regard S as our greedy solution.

Lemma 1. $OPT \geq \frac{1}{2}\sum_{1\leq i\leq n} t_i$

Proof. According to the definition, the total time to complete all jobs is the maximum over the two machines of the total time of all jobs scheduled on the machine, thus, at least we need $\frac{1}{2}\sum_{1\leq i\leq n} t_i$. Hence, the optimal solution is the one that all jobs distributed as evenly as possible over the two machines.

Lemma 2. When the jobs are in decreasing order, we regard the last job to be assigned as n, with smallest time requirement t_n. Then, $t_n \leq \frac{1}{3}OPT$.

Proof. For $n < 5$, we can prove $t_n \leq \frac{1}{3}OPT$ case by case. For $n \geq 5$, we prove that t_n cannot be greater than $\frac{1}{3}OPT$. If $t_n > \frac{1}{3}OPT$, since according to our definition, $t_1 > t_2 > \cdots > t_n$, then there should be at most 2 jobs on each machine. However, with $n \geq 5$, it is impossible.

Lemma 3. $S \leq \frac{1}{2}\sum_{i=1}^{n-1} t_i + t_n$

Proof. The latest starting time of job n is $\frac{1}{2}\sum_{i=1}^{n-1} t_i$,s since this is the time job n will start if all two machines would take equally long in processing the first $n-1$ jobs. If they do not take equally long then some machine becomes available already earlier for starting job n. Thus, it's proved.

After proven Lemma 1, 2 & 3, we can get

$$\begin{aligned} S &\leq \frac{1}{2}\sum_{i=1}^{n-1} t_i + t_n = \frac{1}{2}\sum_{i=1}^{n} t_i + \frac{1}{2}t_n \\ &\leq OPT + \frac{1}{2}t_n \\ &\leq OPT + \frac{1}{2}\cdot\frac{1}{3}OPT \\ &\leq \frac{7}{6}OPT \end{aligned}$$

The approximation ratio for the greedy heuristic is 7/6.

Problem 4: Maximum coverage

Problem Description

The maximum coverage problem is the following: Given a universe U of n elements, with nonnegative weights specified, a collection of subsets of $U, S_1, \ldots, S_l$, and an integer k, pick k sets so as to maximize the weight of elements covered. Show that the obvious algorithm, of greedily picking the best set in each iteration until k sets are picked, achieves an approximation factor of $\left(1-(1-1/k)^k\right) > (1-1/e)$.

Solution

(Proof)

We use OPT to denote the optimal set of solution of the maximum coverage problem at k iteration, let A_i denote the set of newly covered elements at the i_{th} iteration, B_i as the total set of covered elements up to i_{th} iteration, and C_i as the set of elements covered in OPT but uncovered in B_i after the i_{th} iteration.

Let W_S denote the weight of elements covered by set S. Hence, according to the definition, we know that

$$W_{B_i} = \sum_{k,\text{ where } b_k \in B_i} w_{b_k} = W_{B_{i-1}} + \sum_{k,\text{ where } a_k \in A_i} w_{a_k} = W_{B_{i-1}} + W_{A_i}$$

$$W_{OPT} = W_{B_i} + W_{C_i}$$

The base case is $W_{A_0} = W_{B_0} = 0$ and $W_{C_0} = W_{OPT}$.

We prove the greedy algorithm achieves an approximation factor of $\left(1-(1-1/k)^k\right) > (1-1/e)$ by proving the following lemmas:

Lemma 1. $W_{A_{i+1}} \geq W_{C_i}/k$

Proof. According to our definition, we already know that the optimal solution should cover W_{OPT} elements at k iterations. This means at each iteration i, there should be some subsets of U whose weights are greater than or equal to the $1/k$ of the remaining weights of uncovered elements, that is W_{C_i}/k. Otherwise, it was impossible to arrive at the optimal solution at k iterations.

Lemma 1. $W_{C_{i+1}} \leq (1-1/k)^{i+1} \cdot W_{OPT}$

Proof. We will prove this by induction. We first show that this claim is true for base case $i = 0$.

$$\begin{aligned} WC_1 \leq (1-\frac{1}{k}) \cdot W_{OPT} &= W_{OPT} - W_{OPT} \cdot \frac{1}{k} \\ W_{OPT} - W_{B_1} &\leq W_{OPT} - W_{OPT} \cdot \frac{1}{k} \\ W_{B_1} &\geq \frac{1}{k} W_{OPT} \\ W_{A_1} &\geq \frac{1}{k} W_{OPT} \\ W_{A_1} &\geq \frac{1}{k} W_{C_0} \end{aligned}$$

The base case is proved by lemma 1. Then, the inductive hypothesis assume the i iteration is true. We want to prove that the $i+1$ iteration is true. We prove by the following statements:

$$\begin{aligned} W_{C_{i+1}} &= W_{C_i} - W_{A_{i+1}} \\ W_{C_{i+1}} &\leq W_{C_i} - W_{C_i}/k \\ W_{C_{i+1}} &\leq (1-\frac{1}{k}) W_{C_i} \\ W_{C_{i+1}} &\leq (1-\frac{1}{k}) W_{OPT} (1-\frac{1}{k})^i \\ W_{C_{i+1}} &\leq W_{OPT} (1-\frac{1}{k})^{i+1} \end{aligned}$$

It's proved.

By lemma 2, we know that $W_{C_k} \leq (1-\frac{1}{k})^k \cdot W_{OPT}$.

Mathematically, $(1 - \frac{1}{k})^k \approx \frac{1}{e}$, so we can get $W_{C_k} \leq \frac{1}{e} \cdot W_{OPT}$.

$$W_{B_k} = W_{OPT} - W_{C_k}$$
$$W_{B_k} \geq W_{OPT} - \frac{1}{e} \cdot W_{OPT} = (1 - \frac{1}{e})W_{OPT}$$

Thus, we can conclude that the greedy algorithm achieves an approximation factor of $\left(1 - (1 - 1/k)^k\right) > (1 - 1/e)$.