CS60007 Algorithm Design and Analysis 2018
Solution of Assignment 4

Palash Dey and Swagato Sanyal
Indian Institute of Technology, Kharagpur

Please submit the solutions of problem 1, 5, and 12. The deadline is November 12, 2018 in the class.

General Instructions

- ▷ Please prove correctness of every algorithm you design.
- ▷ If not stated otherwise, please ensure that your algorithm runs in polynomial time.
- ▷ If not stated otherwise, please design a deterministic algorithm.
- ▷ If not stated otherwise, please assume that n to be the size (the number of bits) of the input instance.
- ▷ If not stated otherwise, please assume that the graphs under consideration are finite, weighted, and directed graphs which does not contain any self loop.
- ▷ We have not defined some standard problems. You are expected to find the problem statements (from Google say) by yourself.

---

1. Let n denotes the number of vertices in the graph. Prove that if there exists an k^n factor approximation algorithm for the Travelling Salesman problem for any positive integer k, then $\mathsf{P} = \mathsf{NP}$.

   Proof. Let k be any positive integer. We will show that if there exists a k^n factor approximation algorithm for the Travelling Salesman problem, then there exists a polynomial time algorithm for the Hamiltonian Cycle problem on undirected graphs. Let us assume that there exists a k^n factor approximation algorithm $\mathcal{A}$ for the Travelling Salesman problem.

Let $\mathcal{G}$ be an instance of the Hamiltonian Cycle problem. We construct another graph $\mathcal{H}$ from $\mathcal{G}$ as follows. We have $\mathcal{V}[\mathcal{H}] = \mathcal{V}[\mathcal{G}]$; that is, the set of vertices in $\mathcal{H}$ is the same as in $\mathcal{G}$. The graph $\mathcal{H}$ is a complete graph. For any two vertices $u, v \in \mathcal{V}[\mathcal{H}]$ with $u \neq v$, we define the weight $w(\{u, v\})$ of the edge $\{u, v\}$ in $\mathcal{H}$ to be 1 if $\{u, v\} \in \mathcal{E}[\mathcal{G}]$; otherwise we define the weight $w(\{u, v\})$ of the edge $\{u, v\}$ in $\mathcal{H}$ to be n^2k^n. We now claim that there exists a Hamiltonian cycle in $\mathcal{G}$ if and only if there exists a Travelling Salesman tour in $\mathcal{H}$ of weight n. To prove this, in one direction, let us assume that there exists a Hamiltonian cycle $\mathcal{C}$ in $\mathcal{G}$. Then the weight of the tour $\mathcal{C}$ in $\mathcal{H}$ is n. For the other direction, let us assume that there exists a Travelling Salesman tour $\mathcal{T}$ in $\mathcal{H}$ of weight n. Then $\mathcal{T}$ forms a Hamiltonian cycle in $\mathcal{G}$ since if any edge in $\mathcal{T}$ is not present in $\mathcal{G}$, then the weight of the tour $\mathcal{T}$ in $\mathcal{H}$ would be at least n^2k^n which is strictly greater than n for any positive integer k. This proves the claim. Moreover, from the proof of the claim it also follows that if there is no Hamiltonian cycle in $\mathcal{G}$ then the weight of any Travelling Salesman tour in $\mathcal{H}$ is at least n^2k^n.

Since n^2k^n can be computed in polynomial in n time, the graph $\mathcal{H}$ can be constructed in polynomial in n time from $\mathcal{G}$. We run the algorithm $\mathcal{A}$ on $\mathcal{H}$ and output that $\mathcal{G}$ has a Hamiltonian Cycle if and only if $\mathcal{A}$ outputs a tour of weight at most nk^n. We now prove correctness of our algorithm.

If $\mathcal{G}$ has a Hamiltonian cycle, then it follows from the above claim that $\mathcal{H}$ has a Travelling Salesman tour of weight n. Hence, by the definition of an approximation factor, $\mathcal{A}$ returns a Travelling Salesman tour in $\mathcal{H}$ of weight at most nk^n and hence, in the case when $\mathcal{G}$ has a Hamiltonian cycle, our algorithm outputs correctly. If $\mathcal{G}$ does not have any Hamiltonian cycle, then it follows from the above claim that any Travelling Salesman tour in $\mathcal{H}$ has weight at least n^2k^n. In particular, the Travelling Salesman tour returned by the algorithm $\mathcal{A}$ in this case has weight at least n^2k^n which is strictly greater than nk^n. Thus our algorithm correctly outputs also in the case when $\mathcal{G}$ does not have any Hamiltonian cycle. This concludes the proof of correctness of our algorithm. Our algorithm runs in polynomial time since $\mathcal{H}$ can be constructed form $\mathcal{G}$ in polynomial time and the algorithm $\mathcal{A}$ runs in polynomial time. Since the Travelling Salesman problem is NP-complete, we have P = NP. □

2. Design a $\frac{7}{8}$ factor approximation algorithm for the MAX3SAT problem. [Hint: try greedy approach.]

3. Design a $\mathcal{O}(\log n)$ factor approximation algorithm for the Set Cover problem. [Hint: try greedy approach.]

4. Design a 2 factor approximation algorithm for the Vertex Cover problem. [Hint: try to find polynomial time computable lower bound on the size of a minimum vertex cover.]

5. Design a 2 factor approximation algorithm for the Bin Packing problem. [Hint: This is Problem 35-1 in the CLRS book. Follow the guideline provided in CLRS.]

Proof. Let $s_i, i \in [n]$ with $s_i \in (0,1)$ be the set of objects. Let $\mathcal{S} = \sum_{i=1}^{n} s_i$. Since the capacity of every bin is 1, we have OPT $= \lceil \mathcal{S} \rceil$. We use the first fit algorithm. Concretely, we iteratively pick objects s_i for $i \in [n]$ (in any order) and if there exists a bin which can hold s_i (that is the sum of weights of the objects the bin is currently holding is at most $1 - s_i$), then we put s_i to that bin; otherwise we put s_i is a new empty bin. We claim that there is at most one bin which holds a total weight of at most $\frac{1}{2}$ during the run of the algorithm. The claim is clearly true after assigning the first item (since the number of bins is 1 then). Suppose not, then let us assume that the first time when the claim does not hold is after putting the object s_ℓ. Since the claim was true before putting the object s_ℓ, the object must have been put in a new bin and the new bin must contain objects of weight at most $\frac{1}{2}$; that is $s_\ell \leqslant \frac{1}{2}$ since s_ℓ is the only object in the new bin. Since s_ℓ has been assigned in a new bin and $s_\ell \leqslant \frac{1}{2}$, every bin except the new bin must contain objects of total weight strictly more than $\frac{1}{2}$ (otherwise, if there was a bin with total weight at most $\frac{1}{2}$, then our algorithm would have assigned s_ℓ to that bin and never acquired a new bin). We now claim that the number of bin used by the first fit algorithm, denoted by ALG, is at most $\lceil 2\mathcal{S} \rceil$. If $2\mathcal{S}$ is a integer, then it follows from the claim above that ALG $\leqslant 2\mathcal{S}$. On the other hand, if $2\mathcal{S}$ is not an integer, then we have ALG $\leqslant \lfloor 2\mathcal{S} \rfloor + 1 \leqslant \lceil 2\mathcal{S} \rceil$ from the claim above. Since $\lceil 2\mathcal{S} \rceil \leqslant 2\lceil \mathcal{S} \rceil \leqslant 2$OPT, the first fit algorithm is a 2 factor approximation algorithm. □

6. Design a FPTAS for the 0/1-Knapsack problem.

7. Prove that if there exists an α factor approximation algorithm for the Maximum Clique problem, then there exists a $\sqrt{\alpha}$ factor approximation algorithm for the Maximum Clique problem for any constant α. Deduce from this that if if there exists an α factor approximation algorithm for the Maximum Clique problem, then there exists a PTAS for the Maximum Clique problem. [Hint: Use hint from Problem 35-2 in the CLRS book.]

8. Prove that if there exists a FPTAS for the Vertex Cover problem, them P = NP.

9. Design a randomized $\frac{1}{2}$ factor approximation algorithm for the Maximum Cut problem. Deduce from this that for every graph, there exists a cut consisting of at least half the edges in the graph.

10. Design a deterministic $\frac{1}{2}$ factor approximation algorithm for the Maximum Cut problem. [Hint: try greedy approach.]

11. We know that CNF-SAT is NP-complete due to the famous Cook-Levin Theorem. Design a polynomial time algorithm for DNF-SAT.

12. We know that 3SAT is NP-complete. Design a polynomial time algorithm for 2SAT.

    Refer to https://www.iitg.ac.in/deepkesh/CS301/assignment-2/2sat.pdf