Homework Exercises on Approximation Algorithms

The maximum number of points for all exercises is 30. The grade for this homework set is: (number of scored points)/3.

Exercise Set Approx I

A.I-1 (1.5 + 1.5 points) Consider the Load Balancing problem on two machines. Thus we want to distribute a set of n jobs with processing times $t_1, \ldots, t_n$ over two machines such that the makespan (the maximum of the processing times of the two machines) is minimized. Professor Smart has designed an approximation algorithm Alg for this problem, and he claims that his algorithm is a 1.05-approximation algorithm. We run Alg on a problem instance where the total size of all the jobs is 200, and Alg returns a solution whose makespan is 120.

(i) Suppose that we know that all job sizes are at most 100. Can we then conclude that professor Smart's claim is false? Explain your answer.

(ii) Same question when all job sizes are at most 10.

A.I-2 (3 points) Consider a company that has to schedule jobs on a daily basis. That is, each day they get a number of jobs that they schedule for that day on one of their machines. They do the scheduling using the *Greedy-Scheduling* algorithm as described in the Course Notes. (Thus, each day they run *Greedy-Scheduling* on the set of jobs that must be executed on that day.) The following information is known about the company and the jobs: the company has 6 machines, the processing times t_i of the jobs are always between 1 and 25, and the total processing time of all the jobs, $\sum_{i=1}^{n} t_i$, is always at least 1000.

(i) We know from Theorem 1.5 in the Course Notes that *Greedy-Scheduling* is a (11/6)-approximation algorithm. Under the given conditions a stronger result is possible: prove that *Greedy-Scheduling* is a ρ-approximation for some $\rho < 11/6$. Try to make ρ as small as possible.

(ii) Give an example of a set of jobs satisfying the condition stated above such that the makespan produced by *Greedy-Scheduling* on this input is ρ' times the optimal makespan. Try to make ρ' as large as possible.

Note: ideally, the value for ρ' that you prove in (ii) is equal to the value for ρ that you prove in (i). If this is the case, your analysis is *tight*—it cannot be improved—and the value is *the* approximation ratio of the algorithm.

A.I-3 (1+1+1+1 points) Theorem 1.7 in the Course Notes states that *Ordered-Scheduling* is a (3/2)-approximation algorithm for any number of machines. In this exercise you are asked to prove a better bound for the special case $m = 2$. (So, for the rest of the exercise, we assume $m = 2$.)

(i) Prove that if $n \leqslant 4$—that is, if the number of jobs is at most four—then *Ordered-Scheduling* is optimal.

(ii) Prove that for $n = 5$ the algorithm always gives a makespan that is at most $(7/6) \cdot \text{OPT}$, and give an example showing that this bound is tight (that is, an example for $n = 5$ in which *Ordered-Scheduling* gives a makespan that is equal to $(7/6) \cdot \text{OPT}$).

(iii) Following the proof of Theorem 1.7, let M_{i^*} be a machine determining the makespan, and let j^* be the last job assigned to M_{i^*} by *Ordered-Scheduling*. Prove that then the produced makespan is at most $(1 + \frac{1}{j^*}) \cdot \text{OPT}$.

(iv) Prove that the approximation ratio for *Ordered-Scheduling* is 7/6.

Exercise Set Approx II

A.II-1 (1+1+2.5 points) Let $X = \{x_1, \ldots, x_n\}$ be a set of n boolean variables. A boolean formula over the set X is a CNF formula—in other words, is in *conjunctive normal form*—if it has the form $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_j is the disjunction of a number of literals. In this exercise we consider CNF-formulas where every clause has exactly three literals, and there are no negated literals. An example of such a formula is

$$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_7) \wedge (x_1 \vee x_5 \vee x_6).$$

Such CNF formulas are obviously satisfiable: setting all variables to TRUE clearly makes every clause TRUE. Our goal is to make the CNF formula TRUE by setting the smallest number of variables to TRUE.

(i) Consider the following algorithm for this problem.

Greedy-CNF$(\mathcal{C}, X)$
1: $\triangleright$ $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a set of clauses, $X = \{x_1, \ldots, x_n\}$ a set of variables.
2: **while** $\mathcal{C} \neq \emptyset$ **do**
3: Take an arbitrary clause $C_j \in \mathcal{C}$.
4: Let x_i be one of the variables in C_j.
5: Set $x_i \leftarrow$ TRUE.
6: Remove all clauses from $\mathcal{C}$ that contain x_i.
7: **end while**
8: **return** X

Prove or disprove: *Greedy-CNF* is a 4-approximation algorithm.

(ii) Modify the algorithm such that it becomes a 3-approximation algorithm for the problem, and prove that your algorithm achieves the desired approximation ratio. Make sure you explicitly use a lower bound in your proof.

(iii) Now give a different 3-approximation algorithm for the problem, based on the technique of LP relaxation. Prove that your algorithm returns a valid solution, and prove that the algorithm is indeed a 3-approximation algorithm.

A.II-2 (1.5 points) Give an example of an input graph G such that the integrality gap of the LP in *ApproxWeightedVertexCover* is (at least) $2 - \frac{2}{|V|}$. *Hint:* Use a graph where all vertex weights are 1.

A.II-3 (1+3 points) An electricity company has to decide how to connect the houses in a new neighborhood to the electricity network. Connecting a house to the network is done via a *distribution unit*. There are several possible locations where distribution units can be placed. Thus the problem faced by the company is to decide which of the potential distribution units to actually build, and then through which of these units each house will be served. An additional difficulty is that for each house only some of the distribution units are suitable.

This problem can be modeled as follows. We have a set $U = \{u_1, \ldots, u_n\}$ of potential distribution units, each of which has a cost f_i associated to it; the cost f_i must be paid if the company decides to build unit u_i. Moreover, we have a set $H = \{h_1, \ldots, h_m\}$ of houses that need to be connected to the network. Each house has a set $U(h_j) \subseteq U$ of suitable distribution units, and for each $u_i \in U(h_j)$ there is a cost $g_{i,j}$ that must be paid if the company decides to connect house h_j to unit u_i. The goal of the company is to minimize its total cost, which is the cost of building distribution units plus the cost of connecting each house to one of the distribution units.

(i) Formulate the problem as a 0/1 linear program, and briefly explain your formulation.

(ii) Assume that $|U(h_j)| \leqslant 4$ for all h_j. Give a polynomial-time approximation algorithm for this case, based on the technique of LP rounding. Prove that your algorithm returns a valid solution and prove a bound on its approximation ratio.

Exercise Set Approx III

A.III-1 (1 + 2 + 1 points) The TSP problem on a set P of points in the plane is to compute a shortest tour visiting all the points in P, that is, a tour whose (Euclidean) length is minimized.

Suppose we have an algorithm *IntegerTSP*(P) that, given a set P of n points in the plane with integer coordinates in the range $0, \ldots, m$, computes a shortest tour in $O(nm)$ time. Consider the following PTAS for the general TSP problem, that is, for the case where the coordinates need not be integral and we do not have a pre-specified range in which the coordinates lie. We assume that $\min_{p \in P} p_x = \min_{p \in P} p_y = 0$, where p_x and p_y denote the x- and y-coordinate of the point p.

PTAS-TSP(P)
1: $\Delta \leftarrow \ldots$
2: For each point $p \in P$ define $p^* = (p_x^*, p_y^*)$, where $p_x^* = \lceil p_x/\Delta \rceil$ and $p_y^* = \lceil p_y/\Delta \rceil$. Let $P^* := \{p^* : p \in P\}$.
3: Compute a shortest tour on P^* using the algorithm *IntegerTSP*(P^*), and return the reported tour (with each point $p^* \in P^*$ replaced with its corresponding point $p \in P$.

(i) Derive a suitable value to be used for Δ in Step 1, so that the resulting algorithm is a PTAS. (Note: In this part of the exercise you don't have to prove that the algorithm is a PTAS.)

(ii) For a tour T on P, define *length*(T) to be the Euclidean length of T. Moreover, define *length*$^*(T)$ to be the length of T if each point $p \in P$ is replaced by p^*. Let T^* be the tour computed by *PTAS-TSP* and let T_{OPT} be an optimal tour for the set P. Prove that *length*$(T^*) \leqslant (1 + \varepsilon) \cdot$ *length*(T_{OPT}) for your choice of Δ, using a proof similar to the proof of Theorem 3.3 in the Course Notes.

(iii) Analyze the running time of the algorithm for your choice of Δ.

A.III-2 (3 points) Let $\mathcal{G} = (V, E)$ be a connected, undirected graph. An *edge cover* for $\mathcal{G}$ is a subset $C \subseteq E$ of the edges such that each vertex $v \in V$ is incident to at least one edge in C. Let each edge $e = (u, v) \in E$ have a positive weight $w(e) \in \mathbb{R}^+$. We want to find an edge cover for $\mathcal{G}$ whose total weight is minimized.

Suppose that we can solve the edge-cover problem optimally in $O(2^W(|V| + |E|))$ time if the weights are integers in the range $\{1, \ldots, W\}$. Give a PTAS for the case where the weights are real numbers in the range $[1, 10]$. Prove that your algorithm achieves the required approximation ratio and analyze its running time.

A.III-3 (2+1 points) Vertex Cover is NP-complete, so there is no polynomial-time algorithm that solves Vertex Cover optimally unless P=NP.

(i) Prove that this implies that there is no FPTAS for Vertex Cover unless P=NP. (NB: You are not allowed to use the fact mentioned in the Course Notes that Vertex Cover cannot be approximated to within a factor 1.3606 unless P=NP; your proof that an FPTAS does not exist should only be based on the fact that Vertex Cover is NP-complete.)

Hint: Assume Alg(G, ε) is an FPTAS that computes a $(1 + \varepsilon)$-approximation for Vertex Cover on a graph G. Now give an algorithm that solves Vertex Cover exactly by picking a suitable ε and using Alg(G, ε) as a subroutine. Argue that your choice of ε leads to an exact solution and argue that the resulting algorithm runs in polynomial time to derive a contradiction to the existence of an FPTAS.

(ii) Does your proof also imply that there is no PTAS for Vertex Cover unless P=NP? Explain your answer.