CS60007 Algorithm Design and Analysis 2018
Supplemental Material for Approximation Algorithms

Palash Dey
Indian Institute of Technology, Kharagpur

November 5, 2018

Warning: This is a preliminary version and may contain error. If you find any error, please report it to me.

---

1 Introduction

Assuming $\mathsf{P} \neq \mathsf{NP}$, there does not exist any polynomial time algorithm for any NP-complete problem. There can be different approaches to tackle it. For example, (i) one could design various heuristics which will output reasonable solution in quick (polynomial) time (ii) one could design polynomial time approximation algorithms which is guaranteed to output an "approximate" solution (iii) one could design polynomial time algorithms which will output exact solution (that is, solves the problem exactly; not approximately) for certain range of parameters (for example, there exists a polynomial time algorithm for the VERTEX COVER problem if the vertex cover we are looking for is of size at most $\mathcal{O}(\log n)$ where n is the number of vertices in the graph). In this material, we will discuss on the approach of finding an approximate solution. The notion of an approximation factor will be central to our exposition.

Definition 1 (Approximation Factor[1] for a Minimization Problem)**.** *Let $\mathcal{P}$ be a minimization problem and $\alpha : \mathbb{N} \longrightarrow \mathbb{R}_{\geqslant 1}$ be any function from the set of natural numbers to the set of real numbers with value at least 1. An algorithm $\mathcal{A}$ for the problem $\mathcal{P}$ is called an α factor approximation algorithm if, for every instance of Π of $\mathcal{P}$ of "parameter" n, we have ALG(Π) $\leqslant \alpha(n)$OPT(Π) where ALG(Π) and OPT(Π) are the values of the output of the algorithm and optimal solution for the instance Π respectively.*

Similarly, we can define approximation factor for a maximization problem.

Definition 2 (Approximation Factor for a Maximization Problem)**.** *Let $\mathcal{P}$ be a maximization problem and $\alpha : \mathbb{N} \longrightarrow [0, 1]$ be any function from the set of natural numbers to the set of real numbers with values in $[0, 1]$. An algorithm $\mathcal{A}$ for the problem $\mathcal{P}$ is called an α factor approximation algorithm if, for every instance of Π of $\mathcal{P}$ of "parameter" n, we have ALG(Π) $\geqslant \alpha(n)$OPT(Π) where ALG(Π) and OPT(Π) are the values of the output of the algorithm and optimal solution for the instance Π respectively.*

In the above definitions, the notion of the parameter depends on the problem at hand. For example, for the 3−SAT problem, the parameter could be the number of variables or the number of clauses or the maximum number of clauses any variable appears etc.; for the VERTEX COVER problem, the parameter could

[1] Also called approximation ratio.

be the number of vertices or the number of edges or the diameter of the graph or the maximum degree of any node in the graph etc.

2 Approximation Algorithms

In this section, we will see some concrete examples of approximation algorithms. In principle, design of approximation algorithm is more along the line of design polynomial time algorithms– here too, our goal is to design a polynomial time algorithm which approximates the optimal solution well (since we usually design approximation algorithms for NP-hard problems, we do not hope to have an algorithm which solves the problem exactly). One of the starting point in designing polynomial time exact algorithm[2] is to study the structure of an optimal solution to find a "foothold" which can be exploited to come up with a polynomial time algorithm. Similarly, one of the starting point for designing approximation algorithm is to study the structure of an optimal solution to find a good lower bound if the problem is a minimization problem (a good upper bound if the problem is a maximization problem) which can be exploited to come up with an approximation algorithm. Let us see few concrete examples.

2.1 Approximation Algorithm for TRAVELLING SALESMAN PROBLEM

From CLRS.

2.2 Randomized Approximation Algorithm for EXACT3−SAT

In the EXACT3−SAT problem, we are given a collection of CNF clauses each is a logical OR of 3 distinct literals over a set of variables. The objective is to find a Boolean assignment of variables which satisfies a maximum number of clauses. Formally, the problem is defined as follows.

EXACT3−SAT
Input: A collection $\{C_j : j \in [m]\}$ of m clauses each is a logical OR of exactly 3 different literals over a set $\{x_i : i \in [n]\}$ of n Boolean variables.
Output: Output a Boolean assignment of variables which satisfies a maximum number of clauses.

Our randomized algorithm does the following: for each $i \in [n]$, assign x_i to be TRUE or FALSE with equal probability independent of everything else. That is, we toss a fair coin; assign x_i to TRUE if the coin comes up head; otherwise we assign x_i to FALSE. We claim that the expected number of clauses satisfied by the assignment output by the algorithm is $\frac{7}{8}m$.

Lemma 3. *The expected number of clauses satisfied by the assignment output by the algorithm is $\frac{7}{8}m$ which is at least $\frac{7}{8}$OPT(Π) where OPT(Π) is the optimal number of clauses satisfied in an instance Π.*

Proof. Let $f : x_i : i \in [n] \longrightarrow \{\text{TRUE}, \text{FALSE}\}$ be the assignment output by the algorithm (which depends on the randomness used by the algorithm). We define a random variable Y_j for $j \in [m]$ to be 1 if f satisfies the clause C_j and 0 otherwise. Since each C_j is the logical OR of exactly 3 different literals, the random variable Y_j is distributed as follows for every $j \in [m]$[3].

$$Y_j = \begin{cases} 1 & \text{with probability at least } \frac{7}{8} \\ 0 & \text{with probability at most } \frac{1}{8} \end{cases}$$

[2] An algorithm for some problem is called exact if the algorithm finds exact solution of every instance of the problem.
[3] Do you see why the probability of $Y_j = 1$ is not exactly $\frac{7}{8}$?

Thus we have $\mathbb{E}[Y_j] \geqslant \frac{7}{8}$. We also define a random variable Z denoting the number of clauses satisfied by f. Then we have the following.

$$Z = \sum_{j=1}^{m} Y_j$$
$$\Rightarrow \mathbb{E}[Z] = \sum_{j=1}^{m} \mathbb{E}[Y_j] \geqslant \sum_{j=1}^{m} \frac{7}{8} = \frac{7}{8}m$$

□

The following structural result immediately follows from Lemma 3. The argument uses the well known probabilistic method.

Corollary 4. *In any* EXACT3−SAT *instance, there exists an assignment which satisfies at least* $\frac{7}{8}$ *fraction of the number of clauses.*

Proof. In the proof of Lemma 3, we have seen that $\mathbb{E}[Z] \geqslant \frac{7}{8}m$. On the other hand, we have the following expression for $\mathbb{E}[Z]$. As defined in the proof of Lemma 3, f is the assignment output by the algorithm.

$$\mathbb{E}[Z] = \sum_{\ell=1}^{m} \Pr\left[f \text{ satisfies exactly } \ell \text{ clauses}\right] \times \ell$$

For the sake of finding a contradiction, let us assume that there exists an instance Π of EXACT3−SAT for which there does not exist any assignment which satisfies at least $\frac{7}{8}$ fraction of the number of clauses in Π, then we have the following which is a contradiction.

$$\frac{7}{8}m \leqslant \mathbb{E}[Z] = \sum_{\ell=1}^{m} \Pr\left[f \text{ satisfies exactly } \ell \text{ clauses}\right] \times \ell = \sum_{\ell=1}^{(\frac{7}{8}m)-1} \Pr\left[f \text{ satisfies exactly } \ell \text{ clauses}\right] \times \ell < \frac{7}{8}m$$

The second equality follows from the assumption that there does not exist any assignment in Π which satisfies at least $\frac{7}{8}$ fraction of the number of clauses (hence the probability that the assignment f satisfies at least $\frac{7m}{8}$ clauses is 0). □

2.3 Notion of PTAS, EPTAS, and FPTAS

One of the primary goal of approximation algorithm design is to obtain an algorithm with approximation factor as close to 1 as possible. Notice that, a polynomial time approximation algorithm with approximation factor 1 is equivalent to having an exact polynomial time algorithm for the problem which we do not hope to achieve for an NP-complete problem (since that would imply $P = NP$). So it would be excellent if we have a polynomial time approximation with approximation factor $(1+\varepsilon)$ for a minimization problem ($(1-\varepsilon)$ for a maximization problem) for any arbitrary constant $\varepsilon > 0$. The notion of PTAS formalizes this concept.

Definition 5 (Polynomial Time Approximation Scheme (PTAS))**.** *An algorithm is called a polynomial time approximation scheme (PTAS for short) for a problem if it takes a parameter* $\varepsilon > 0$ *as input in addition to the usual input and outputs a solution with approximation factor* $(1+\varepsilon)$ *for a minimization problem and* $(1-\varepsilon)$ *for a maximization problem in time* $O(n^{f(1/\varepsilon)})$ *where* n *is the length of input and* $f(1/\varepsilon)$ *is any computable function.*

For example, the Euclidean Travelling Salesman problem where the input is a set of n points (which are the vertices) in a d-dimensional Euclidean space for some constant d and the distance between vertices are the corresponding Euclidean distances and one need to find a tour of minimum weight which covers every vertex, admits a PTAS – the running time of the algorithm is $O(n(\log n)^{(O(\sqrt{d}/\varepsilon))^{d-1}})$. As we can immediately observe that the running time of a PTAS can increase rapidly as we decrease ε. It would be better if ε does not appear in the exponent of n in the running time of our algorithm. EPTAS formalizes this notion.

Definition 6 (Efficient Polynomial Time Approximation Scheme (EPTAS))**.** *An algorithm is called an efficient polynomial time approximation scheme (EPTAS for short) for a problem if it takes a parameter $\varepsilon > 0$ as input in addition to the usual input and outputs a solution with approximation factor $(1+\varepsilon)$ for a minimization problem and $(1-\varepsilon)$ for a maximization problem in time $f(1/\varepsilon)O(n^{O(1)})$ where n is the length of input and $f(1/\varepsilon)$ is any computable function.*

For example, the Feedback Vertex Set problem on planar graphs where the input is a planar directed graph and we need to find a minimum set of vertices whose removal makes the graph acyclic, admits a EPTAS. Still the dependence on ε in case of an EPTAS can be arbitrarily bad; for example an algorithm with running time $2^{2^{2^{1/\varepsilon}}}O(n)$ is an EPTAS. However, the running time grows rapidly when we decrease ε. To address this, the notion of FPTAS has been proposed.

Definition 7 (Fully Polynomial Time Approximation Scheme (FPTAS))**.** *An algorithm is called an fully polynomial time approximation scheme (FPTAS for short) for a problem if it takes a parameter $\varepsilon > 0$ as input in addition to the usual input and outputs a solution with approximation factor $(1+\varepsilon)$ for a minimization problem and $(1-\varepsilon)$ for a maximization problem in time $O(1/\varepsilon)^{O(1)}O(n^{O(1)})$ where n is the length of input.*

For example, the Subset Sum, Knapsack problems admit FPTAS.

2.4 PTAS for SUBSET SUM

From CLRS.

3 Approximation Lower Bounds

In this section we will refute existence of algorithms of certain approximation guarantees.

3.1 No FPTAS for SET COVER

We will prove that the SET COVER problem does not admit an FPTAS assuming $P \neq NP$. In the SET COVER problem, the input is a universe U of size n, a collection C of m subsets of U, and the objective is to find a minimum number of subsets in C whose union is U.

Theorem 8. *There does not exist an FPTAS for the* SET COVER *problem assuming* $P \neq NP$.

Proof. For the sake of finding a contradiction, let us assume that there exists a $(1+\varepsilon)$ factor approximation algorithm A for the SET COVER problem with running time $O(1/\varepsilon)^{O(1)}O(n^{O(1)})$. We set $\varepsilon = \frac{1}{2m}$ in the algorithm A. The running time of the algorithm A with $\varepsilon = \frac{1}{2m}$ is $O(n^{O(1)})$ with is a polynomial in n. By the definition of an approximation factor, if ALG and OPT denote the number of sets returned by the algorithm and the number of sets in an optimal solution respectively, then we have the following.

$$OPT \leqslant ALG \leqslant (1+\varepsilon)OPT = (1+\frac{1}{2m})OPT = OPT + \frac{OPT}{2m} \leqslant OPT + \frac{1}{2}$$

The last inequality follows from the observation that OPT $\leqslant m$. Since both ALG and OPT are integers, the above inequality implies ALG=OPT. Hence the algorithm $\mathcal{A}$ with $\varepsilon = \frac{1}{2m}$ solves the SET COVER problem exactly in polynomial amount of time which contradicts our assumption that $P \neq NP$ since SET COVER is a NP-complete problem. □