Introduction to Approximation Algorithms

1 Approximation algorithms and performance ratios

To date, thousands of natural optimization problems have been shown to be NP-hard [8, 18]. To deal with these problems, two approaches are commonly adopted: (a) approximation algorithms, (b) randomized algorithms. Roughly speaking, approximation algorithms aim to find solutions whose costs are as close to be optimal as possible in polynomial time. Randomized algorithms can be looked at from different angles. We can design algorithms giving optimal solutions in expected polynomial time, or giving expectedly good solutions.

Many different techniques are used to devise these kinds of algorithms, which can be broadly classified into: (a) combinatorial algorithms, (b) linear programming based algorithms, (c) semi-definite programming based algorithms, (d) randomized (and derandomized) algorithms, etc. Within each class, standard algorithmic designs techniques such as divide and conquer, greedy method, and dynamic programming are often adopted with a great deal of ingenuity. Designing algorithms, after all, is as much an art as it is science.

There is often no point approximating anything unless the approximation algorithm runs efficiently, i.e. in polynomial time. (There are, of course, exceptions.) Hence, when we say approximation algorithms we implicitly imply polynomial time algorithms.

To clearly understand the notion of an approximation algorithm, we first define the so-called *approximation ratio*. Informally, for a minimization problem such as the VERTEX COVER problem, a polynomial time algorithm $\mathcal{A}$ is said to be an approximation algorithm with approximation ratio δ if and only if for *every* instance of the problem, $\mathcal{A}$ gives a solution which is *at most* δ times the optimal value for that instance. This way, δ is always at least 1. As we do not expect to have an approximation algorithm with $\delta = 1$, we would like to get δ as close to 1 as possible. Conversely, for maximization problems the algorithm $\mathcal{A}$ must produce, for each input instance, a solution which is at least δ times the optimal value for that instance. (In this case $\delta \leq 1$.)

The algorithm $\mathcal{A}$ in both cases are said to be δ-approximation algorithm. Sometimes, for maximization problems people use the term *approximation ratio* to refer to $1/\delta$. This is to ensure that the ratio is at least 1 in both the $\min$ and the $\max$ case. The terms approximation ratio, approximation factor, *performance guarantee*, *worst case ratio*, *absolute worst case ratio*, are more or less equivalent, except for the $1/\delta$ confusion we mentioned above.

Let us now be a little bit more formal. Consider an optimization problem Π, in which we try to minimize a certain objective function. For example, when Π is VERTEX COVER the objective function is the size of a vertex cover. For each instance $I \in \Pi$, let $\text{OPT}(I)$ be the optimal value of the objective function for I. In VERTEX COVER, I is a graph G and $\text{OPT}(G)$ depends on the structure of G. Given a polynomial time algorithm $\mathcal{A}$ which returns some feasible solution for Π, let $A(I)$ denote the objective value returned by $\mathcal{A}$ on in put I. Define

$$R_{\mathcal{A}}(I) := \frac{A(I)}{\text{OPT}(I)} \text{ (minimization case)}, \tag{1}$$

called the performance ratio of $\mathcal{A}$ on input I. When Π is a maximization problem, we agree that

$$R_{\mathcal{A}}(I) := \frac{\text{OPT}(I)}{A(I)} \text{ (maximization case).} \tag{2}$$

Definition 1.1 (Performance ratio). A function $r : \mathbb{Z}^+ \to \mathbb{R}^+$ is called a *performance ratio* of algorithm $\mathcal{A}$ if $R_{\mathcal{A}}(I) \geq r(n)$ for all instances I with size n. In this case, we refer to $\mathcal{A}$ as an $r(n)$-approximation algorithm.

Definition 1.2 (PTAS). Suppose for each $\epsilon > 0$, we have a $(1+\epsilon)$-approximation algorithm $\mathcal{A}_\epsilon$ for Π, whose running time is polynomial for a fixed ϵ, then this family of algorithms is called a *polynomial time approximation scheme* (PTAS) for Π.

Definition 1.3 (FPTAS). A PTAS is called a *fully polynomial time approximation scheme* (FPTAS) if the running time of $\mathcal{A}_\epsilon$ is polynomial in both the input size and $\frac{1}{\epsilon}$.

For instance, if on each input of size n the algorithm $\mathcal{A}_\epsilon$ runs in time $O(n^{1+1/\epsilon})$ then we have a PTAS, but not a FPTAS. On the other hand, if the running time was $\Theta(n^{100}\epsilon^{-1000})$, then we get an FPTAS.

The definitions sounded nice and all, but how is it possible that we can come up with an algorithm giving solution provably close to be optimal when computing the optimal value is already **NP**-hard (let alone computing an optimal solution)? Let us look at the following algorithm for VERTEX COVER, which asks us to compute a minimum vertex cover of a given graph. Note that a *matching* is a subset of edges of G no two of which share an end point.

APPROX-VERTEX-COVER(G)

1: Arbitrarily find a maximal matching M of G.
2: **return** all vertices in M.

A maximal matching can clearly be computed in polynomial time: pick an arbitrary edge e of G, remove all edges sharing an end point with e, then repeat the process. Moreover, the set of all vertices of a maximal matching covers all edges. The crucial point to notice is the following. Let M be any matching in G. Let $\text{OPT}(G)$ denote the size of a minimum vertex cover of G. Then, $\text{OPT}(G) \geq M$ since each edge of M has to be covered by at least one vertex. In particular, $\text{OPT}(G) \geq M$ when M is a maximal matching returned by the above algorithm. Thus,

$$\text{APPROX-VERTEX-COVER}(G) = 2|M| \leq 2\text{OPT}(G).$$

We have just proved the following theorem.

Theorem 1.4. APPROX-VERTEX-COVER *is a* 2*-approximation algorithm.*

2 Combinatorial Algorithms

The approximation algorithm APPROX-VERTEX-COVER is a perfect introduction to the combinatorial methods of designing approximation algorithms. It is difficult to get a hold of exactly what we mean by "combinatorial methods." Basically, algorithms which make use of discrete structures and ideas (mostly graph theoretic) are often referred to as combinatorial algorithms.

One of the best examples of combinatorial approximation algorithms is a greedy algorithm approximating SET COVER. An instance of the SET COVER problem consists of a universe set $U = \{1, \ldots, n\}$ and a family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U. We want to find a sub-family of $\mathcal{S}$ with as few sets as possible, such that the union of the sub-family is U (i.e. covers U).

An obvious greedy algorithm for this problem is as follows.

GREEDY-SET-COVER$(U, \mathcal{S})$

1: $\mathcal{C} = \emptyset$
2: **while** $U \neq \emptyset$ **do**
3: Pick $S \in \mathcal{S}$ which covers the most number of elements in U
4: $U \leftarrow U - S$
5: $\mathcal{C} = \mathcal{C} \cup \{S\}$
6: **end while**
7: **return** $\mathcal{C}$

Theorem 2.1. *Let $d = \max\{|S| : S \in \mathcal{S}\}$, then* GREEDY-SET-COVER *is an H_d-approximation algorithm, where $H_d = (1 + \frac{1}{2} + \cdots + \frac{1}{d})$ is the dth harmonic number.*

Proof. Suppose GREEDY-SET-COVER returns a cover of size k. For $1 \leq i \leq k$, let X_i be the set of newly covered elements of U after the ith iteration. Note that $X_1 \cup \cdots \cup X_k = U$. Let $x_i = |X_i|$, then x_i is the maximum number of un-covered elements which can be covered by any set after the $(i-1)$th step.

For each element $u \in X_i$, assign to u a cost $c(u) = 1/x_i$. Then, the cost of the solution returned by GREEDY-SET-COVER is $k = \sum_{u \in U} c(u)$. Let $\mathcal{T} \subseteq \mathcal{S}$ be any optimal solution. Then,

$$k = \sum_{u \in U} c(u) \leq \sum_{T \in \mathcal{T}} \sum_{u \in T} c(u), \tag{3}$$

because each element of U is covered by at least one $T \in \mathcal{T}$.

Consider any set $S \in \mathcal{S}$. For $i \in [k]$, let $a_i = |S \cap X_i|$, then it is easy to see that $x_i \geq a_i + \cdots + a_k$. Consequently,

$$\sum_{u \in S} c(u) = \sum_{i=1}^{k} \frac{a_i}{x_i} \leq \sum_{i=1}^{k} \frac{a_i}{a_i + \cdots + a_k} \leq H_{|S|} \leq H_d.$$

Inequality (3) now implies

$$k \leq \sum_{T \in \mathcal{T}} \sum_{u \in T} c(u) \leq |\mathcal{T}| \cdot H_d.$$

□

Exercise 1. The WEIGHTED SET COVER problem is similar to the SET COVER problem, except that every set has a weight defined by a weight function $w : \mathcal{S} \to \mathbb{Z}^+$. The objective is to find a set cover with minimum weight.

1. State the decision version of WEIGHTED SET COVER and show that it is **NP**-complete.
2. Consider the following greedy algorithm:

GREEDY-WEIGHTED-SET-COVER$(U, \mathcal{S}, w)$
1: $\mathcal{C} = \emptyset$
2: **while** $U \neq \emptyset$ **do**
3: Pick $S \in \mathcal{S}$ with the least cost per un-covered element, i.e. pick S such that $w(S)/|S \cap U|$ is minimized.
4: $U \leftarrow U - S$
5: $\mathcal{C} = \mathcal{C} \cup \{S\}$
6: **end while**
7: **return** $\mathcal{C}$

Let X_i be the set of newly covered elements of U after the ith step. Let $x_i = |X_i|$, and w_i be the weight of the ith set picked by the algorithm. Assign a cost $c(u) = w_i/x_i$ to each element $u \in X_i$, for all $i \leq k$. Show that, for any set $S \in \mathcal{S}$,

$$\sum_{u \in S} c(u) \leq w(S) \cdot H_{|S|}.$$

3. Show that GREEDY-WEIGHTED-SET-COVER has approximation ratio H_d, too.

What is amazing is that $H_d \approx \ln d$ is basically the best approximation ratio we can hope for, as was shown by Feige [13].

Exercise 2 (Bin Packing). Suppose we are given a set of n objects, where the size s_i of the ith object satisfies $0 < s_i < 1$. We wish to pack all the objects into the minimum number of unit-size bins. Each bin can hold any subset of the objects whose total size does not exceed 1.

1. Prove that the problem of determining the minimum number of bins required is **NP**-hard. (Hint: use SUBSET SUM).

   The **first fit** heuristic takes each object in turn and places it into the first bin that can accommodate it. Let $S = \sum_{i=1}^{n} s_i$.

2. Show that the optimal number of bins required is at least $\lceil S \rceil$.

3. Show that the first-fit heristic leaves at most one bin less than half full.

4. Show that the number of bins used by the first-fit heuristic is never more than $\lceil 2S \rceil$.

5. Prove that the first-fit heuristic has approximation ratio 2.

6. Give an efficient implementation of the first-fit heuristic, and analyze its running time.

3 Linear and integer linear programming

A *linear program* consists of a linear objective function which we are trying to maximize or minimize subject to a set of linear equalities and inequalities. For instance, the following is a linear program:

$$\begin{array}{lrcrcrcrcr} \min & x_1 & - & x_2 & + & 4x_3 & & & & \\ \text{subject to} & 3x_1 & - & x_2 & & & & & = & 3 \\ & & - & x_2 & & & + & 2x_4 & \geq & 4 \\ & x_1 & & & + & x_3 & & & \leq & -3 \\ & & & & & & & x_1, x_2 & \geq & 0 \end{array}$$

Linear programs can be solved in polynomial time, where the outcome is either an optimal solution or an indication that the problem is either *unbounded* or *infeasible*.

An *integer (linear) program* is similar to a linear program with an additional requirement that variables are integers. The INTEGER PROGRAMMING problem is the problem of determining if a given integer program has a feasible solution. This problem is known to be **NP**-hard. Hence, we cannot hope to solve general integer programs efficiently. However, integer programs can often be used to formulate a lot of discrete optimization problems.

Let us see how we can formulate VERTEX COVER as an integer program. Suppose we are given a graph $G = (V, E)$ with n vertices and n edges. For each $i \in V = \{1, \ldots, n\}$, let $x_i \in \{0, 1\}$ be a

variable which is 1 if i belongs to the vertex cover, and 0 otherwise; then, the problem is equivalent to solving the following (linear) *integer program*:

$$\begin{array}{ll} \min & x_1 + x_2 + \cdots + x_n \\ \text{subject to} & x_i + x_j \geq 1, \quad \forall ij \in E, \\ & x_i \in \{0,1\}, \quad \forall i \in V. \end{array} \tag{4}$$

The objective function basically counts the number of vertices in the vertex cover. Each inequality $x_i + x_j \geq 1$ requires the corresponding edge ij to have at least one of its end points in the vertex cover. Actually, the formulation above is somewhat too strict. Suppose we relax it a little bit:

$$\begin{array}{ll} \min & x_1 + x_2 + \cdots + x_n \\ \text{subject to} & x_i + x_j \geq 1, \quad \forall ij \in E, \\ & x_i \geq 0, x_i \in \mathbb{Z} \quad \forall i \in V. \end{array}$$

Then, this would still be equivalent to solving the vertex cover problem, since in an optimal solution to the integer program above, no x_i is more than 1.

The next problem is a generalized version of the VC problem.

WEIGHTED VERTEX COVER

Given a graph $G = (V, E)$, $|V| = n$, $|E| = m$, a weight function $w : V \to \mathbb{R}$. Find a vertex cover $C \subseteq V$ for which $\sum_{i \in C} w(i)$ is minimized.

Note that when $w \equiv 1$, the weighted version is the same as the non-weighted version. An equivalent linear integer program can be written as

$$\begin{array}{ll} \min & w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \\ \text{subject to} & x_i + x_j \geq 1, \quad \forall ij \in E, \\ & x_i \in \{0,1\}, \quad \forall i \in V. \end{array}$$

Note that if the weights were all non-negative, then we only have to require the variables to be non-negative integers, just like in the case of normal vertex cover. An integer program (IP) as above is also referred to as a 01-integer program.

The next two problems are more general versions of the VERTEX COVER and the WEIGHTED VERTEX COVER problems. Recall that we use $[n]$ to denote the set $\{1, \ldots, n\}$, for all positive integer n, and $[0]$ naturally denotes $\emptyset$.

SET COVER

Given a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ of subsets of $[m] = \{1, \ldots, m\}$. Find a sub-collection $\mathcal{C} = \{S_i \mid i \in J\}$ with as few members as possible (i.e. $|J|$ as small as possible) such that $\bigcup_{i \in J} S_i = [m]$.

Similar to VERTEX COVER, we use a 01-variable x_j to indicate the inclusion of S_j in the cover. For each $i \in \{1, \ldots, m\}$, we need at least one of the S_j containing i to be picked. The integer program is then

$$\begin{array}{ll} \min & x_1 + \cdots + x_n \\ \text{subject to} & \displaystyle\sum_{j : S_j \ni i} x_j \geq 1, \quad \forall i \in [m], \\ & x_j \in \{0,1\}, \quad \forall j \in [n]. \end{array}$$

WEIGHTED SET COVER

Given a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ of subsets of $[m] = \{1, \ldots, m\}$, and a weight function $w : \mathcal{S} \to \mathbb{R}$. Find a cover $\mathcal{C} = \{S_j \mid j \in J\}$ with minimum total weight.

Trivially, we have

$$\begin{array}{ll} \min & w_1x_1 + \cdots + w_nx_n \\ \text{subject to} & \displaystyle\sum_{j:S_j \ni i} x_j \geq 1, \quad \forall i \in [m], \\ & x_j \in \{0,1\}, \quad \forall j \in [n]. \end{array}$$

TRAVELING SALESMAN (TSP)

A salesman must visit n cities each exactly once and return to the originating city. Given the time to go from city i to city j is t_{ij}, find a tour which takes the shortest time.

This problem is slightly more difficult to formulate then the ones we have seen. Let x_{ij} be the 01 variable indicating if the salesman does go from city i to city j. Obviously, we want the salesman to go into i exactly once and to go out of j exactly once for each city i, j. This condition alone, however, does not ensure connectivity as we might form a few disjoint cycles. We also need constraints to ensure the connectivity of our tour. This could be done by checking for each non-empty set $S \subset [n]$ if there was at least one edge leaving S. In summary, we have the following equivalent linear integer program:

$$\begin{array}{ll} \min & \sum_{i \neq j} t_{ij} x_{ij} \\ \text{subject to} & \sum_{j:j \neq i} x_{ij} = 1, \qquad \forall i \in [n], \\ & \sum_{i:i \neq j} x_{ij} = 1, \qquad \forall j \in [n], \\ & \sum_{i \in S, j \notin S} x_{ij} \geq 1, \quad \forall S \subset [n], S \neq \emptyset \\ & x_{ij} \in \{0,1\}, \quad \forall i, j \in [n], i \neq j. \end{array}$$

Exercise 3 (ASSIGNMENT PROBLEM**).** Formulate the following problem as an IP problem:

There are n processors and n tasks. Each processor might be more suitable to perform a particular kind of task. Hence, there is a cost w_{ij} associated if processor i was to do task j. Find a one-to-one assignment of processors to tasks which minimizes the total cost.

Exercise 4 (KNAPSACK**).** Formulate the following problem as an IP problem:

Given n items with values $v_1, \ldots, v_n$, and weights $w_1, \ldots, w_n$, correspondingly. Find a subset S of items with total weight at most a given W, such that the total value of S is as large as possible.

Exercise 5 (INDEPENDENT SET**).** Formulate the following problem as an IP problem:

Given a graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}^+$, find an independent set of maximum total weight; namely, a subset $P \subseteq V$ of vertices such that no pair of vertices in P are adjacent and the sum $\sum_{i \in P} w(i)$ is maximized.

Exercise 6. Given an $m \times n$ matrix A whose entries are either 0 or 1, and $w \in \mathbb{Z}^n, c \geq \vec{0}$. As usual, we use $\vec{0}$ to denote the all-0 vector, whose dimension is implicit in the (in)equality involved, and $\vec{1}$ to denote the all-1 vector. The (weighted) SET COVER problem has the form

$$\min \left\{ w^T x \mid Ax \geq \vec{1}, x \in \{0,1\}^n \right\}, \tag{5}$$

while the INDEPENDENT SET problem in the previous exercise has the form

$$\max \left\{ w^T x \mid Ax \leq \vec{1}, x \in \{0,1\}^n \right\}, \tag{6}$$

(That is, if you do it correctly.)

In this problem, we shall see that the converse is also true:

(i) Given an integer program of the form (5), where A is **any** 01-matrix, formulate a (weighted) SET COVER instance which is equivalent to the program.

(ii) Given an integer program of the form (6), where A is **any** 01-matrix, formulate an INDEPENDENT SET instance which is equivalent to the program.

In both questions, show the "equivalence." For example, in (i) you must show that the minimum weighted set cover of the constructed set family corresponds to an optimal solution of the integer program and vice versa.

Exercise 7 (BIN PACKING**).** Formulate the following problem as an IP problem:

> Given a set of n items $\{1, \ldots, n\}$, and their "size" $s(i) \in (0, 1]$. Find a way to partition the set of items in to a minimum number m of "bins" $B_1, \ldots, B_m$, such that
>
> $$\sum_{i \in B_j} s(i) \leq 1, \quad \forall j \in [m].$$

4 Relaxation and rounding

In general, *relaxation* refers to the action of relaxing the integer requirement of a linear IP to turn it into an LP. For example, the LP corresponding to the IP (4) of VERTEX COVER is

$$\begin{array}{ll} \min & x_1 + x_2 + \cdots + x_n \\ \text{subject to} & x_i + x_j \geq 1, \quad \forall ij \in E, \\ & 0 \leq x_i \leq 1, \quad \forall i \in V. \end{array} \tag{7}$$

Obviously if the LP version is infeasible, then the IP version is also infeasible. This is the first good reason to do relaxation. Now, suppose $\mathbf{x}^*$ is an optimal solution to the LP problem. We know that $\mathbf{x}^*$ can be found in polynomial time. We can construct a feasible solution $\mathbf{x}^A$ to the IP problem as follows. Let

$$x_i^A = \begin{cases} 1 & \text{if } x_i^* \geq 1/2 \\ 0 & \text{if } x_i^* < 1/2. \end{cases}$$

You should check that $\mathbf{x}^A$ is indeed feasible for the IP. This technique of constructing a feasible solution for the IP from the LP is called *rounding*. We have just seen the second advantage of doing relaxation. The third is that an optimal value for the LP provides a lower bound for the optimal value of the IP. Using this fact, one can derive the approximation ratio of the feasible solution $\mathbf{x}^A$. Let $\text{OPT}(IP)$ be the optimal value for the IP instance, $\text{OPT}(LP)$ the optimal value for the LP, and $\text{cost}(x^A)$ the cost of the feasible solution $\mathbf{x}^A$ for the IP, then we have

$$\begin{aligned} \text{OPT}(IP) &\geq \text{OPT}(LP) \\ &= x_1^* + \cdots + x_n^* \\ &\geq \frac{1}{2}x_1^A + \cdots + \frac{1}{2}x_n^A \\ &= \frac{1}{2}\,\text{cost}(x^A). \end{aligned}$$

In other words, the cost of the approximation $\mathbf{x}^A$ is at most twice the optimal. We thus have a 2-approximation algorithm to solve the VERTEX COVER problem. Since it is impossible, unless P = NP, to have an exact polynomial time algorithm to solve VERTEX COVER, an algorithm giving a feasible solution within twice the optimal is quite satisfactory. The exact same technique works for the WEIGHTED

VERTEX COVER problem, when the weights are non-negative. Thus, we also have a 2-approximation algorithm for the WEIGHTED VERTEX COVER problem. (Note, again, that when we say "approximation algorithm," it automatically means a polynomial-time approximation algorithm. There would be little point approximating a solution if it takes exponentially long.)

Theorem 4.1. *There is an approximation algorithm to solve the* WEIGHTED VERTEX COVER *problem with approximation ratio* 2.

To this end, let us attempt to use the relaxation and rounding idea to find approximation algorithms for the WEIGHTED SET COVER problem. In fact, we shall deal with the following much more general problem called the GENERAL COVER problem:

$$\begin{array}{lccccccll} \min & c_1x_1 & + & \dots & + & c_nx_n & & & \\ \text{subject to} & a_{i1}x_1 & + & \dots & + & a_{in}x_n & \geq & b_i, & i \in [m]. \\ & & & & & x_j \in \{0,1\}, & & & \forall j \in [n], \end{array} \tag{8}$$

where a_{ij}, b_i, c_j are all non-negative integers. Because we can remove an inequality if $b_i = 0$, we can assume that $\mathbf{b} > 0$. Moreover, if $c_j = 0$ then we can set $x_j = 1$ and remove the column corresponding to j without effecting the objective function as well as feasible solutions. Thus, we can also assume that $c > 0$. Lastly, assume $\sum_j a_{ij} \geq b_i$, for all $i \in [m]$; otherwise, the system is infeasible.

The relaxed LP version for (8) is

$$\begin{array}{lccccccll} \min & c_1x_1 & + & \dots & + & c_nx_n & & & \\ \text{subject to} & a_{i1}x_1 & + & \dots & + & a_{in}x_n & \geq & b_i, & i \in [m]. \\ & & & & & 0 \leq x_j \leq 1, & & & \forall j \in [n], \end{array} \tag{9}$$

Let $\mathbf{x}^*$ be an optimal solution to the LP version. How would we round $\mathbf{x}^*$ to get $\mathbf{x}^A$, as in the VERTEX COVER case? Firstly, the rounding must ensure that $\mathbf{x}^A$ is feasible, namely they must satisfy each of the inequalities $a_{i1}x_1 + \cdots + a_{in}x_n \geq b_i$. Secondly, we do not want to do "over-rounding," such as assigning everything to 1, which would give a feasible solution but it does not give a very good approximation. Consider an inequality such as

$$3x_1^* + 4x_2^* + x_3^* + 2x_4^* \geq 4, \tag{10}$$

which $\mathbf{x}^*$ satisfies. If we were to round some of the x_i^* up to 1, and the rest down to 0, we must pick the ones whose coefficients sum up to 4 or more; for instance, we could round x_2^* up to 1 and the rest to 0, or x_1^* and x_3^* to 1 and the rest to 0. The difficulty with this idea is that there might be an exponential number of ways to do this, and we also have to do this consistently throughout all inequalities $a_{i1}x_1 + \cdots + a_{in}x_n \geq b_i$. We cannot round x_1^* to 0 in one inequality, and to 1 in another inequality. Fortunately, some information about which x_j^* to round is contained in the actual values of the x_j^*. Consider inequality (10) again. The sum of all coefficients of the x_j^* is 10. If all x_j^* were at most $1/10$, then the left hand side is at most 1. Hence, there must be one x_j^* which is $\geq 1/10$. If $x_2^* \geq 1/10$, and we round it up to 1, then we'd be fine. However, if x_3^* and x_4^* are the only ones which are $\geq 1/10$, then rounding them up to 1 is not sufficient. Fortunately, that cannot happen, since if $x_1^*, x_2^* < 1/10$ and $x_3^*, x_4^* \geq 1/10$, then

$$3x_1^* + 4x_2^* + x_3^* + 2x_4^* < \frac{3}{10} + \frac{4}{10} + 1 + 2 < 4.$$

Thus, *the sum of the coefficients of the x_j^* which are at least $1/(a_{i1} + \cdots + a_{in})$ has to be at least b_i.* This is an informal proof. We shall give a rigorous proof later.

The analysis above leads to the following rounding strategy. Let

$$f = \max_{i=1..n} \left(\sum_{j=1}^{n} a_{ij} \right),$$

and set

$$x_j^A = \begin{cases} 1 & \text{if } x_j^* \geq \frac{1}{f} \\ 0 & \text{if } x_j^* < \frac{1}{f}, \end{cases}$$

then we have an approximation ratio of f. You should map this rounding back to the rounding of the VERTEX COVER problem to see the analogy.

Theorem 4.2. *The rounding strategy above gives an approximation algorithm with approximation ratio at most*

$$f = \max_{i=1..n} \left(\sum_{j=1}^{n} a_{ij} \right)$$

for the GENERAL COVER *problem; namely for every instance of the* GENERAL COVER *problem, the relaxation and rounding algorithm yields a solution of cost at most f times the optimal.*

Proof. We first show that x^A is indeed feasible for IP-GC (the integer program for the GENERAL COVER problem). Suppose x^A is not feasible, then there is some row i for which

$$a_{i1}x_1^A + \cdots + a_{in}x_n^A < b_i.$$

This is the same as

$$\sum_{j:x_j^* \geq 1/f} a_{ij} \leq b_i - 1.$$

If $\sum_{j:x_j^* < 1/f} a_{ij} > 0$, then

$$\sum_{j=1}^{n} a_{ij}x_j^* = \sum_{j:x_j^* \geq 1/f} a_{ij}x_j^* + \sum_{j:x_j^* < 1/f} a_{ij}x_j^* < \sum_{j:x_j^* \geq 1/f} a_{ij} + 1 \leq b_i,$$

which is a contradiction. On the other hand, when $\sum_{j:x_j^* < 1/f} a_{ij} = 0$, we get

$$\sum_{j=1}^{n} a_{ij}x_j^* = \sum_{j:x_j^* \geq 1/f} a_{ij}x_j^* + \sum_{j:x_j^* < 1/f} a_{ij}x_j^* = \sum_{j:x_j^* \geq 1/f} a_{ij} \leq b_i - 1,$$

which again is a contradiction to the feasibility of x^*.

For the performance ratio, notice that

$$\text{cost}(x^A) = \sum_{j=1}^{n} c_j x_j^A \leq \sum_{j=1}^{n} c_j(f x_j^*) = f \text{ OPT(LP-GC)} \leq f \text{ OPT(IP-GC)}.$$

□

Exercise 8. Describe the ratio f for the WEIGHTED SET COVER problem in terms of m, n and the set collection $\mathcal{S}$.

5 Randomized algorithms

A *conjunctive normal form* (CNF) formula is a boolean formula on n variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ consisting of m *clauses* $C_1, \ldots, C_m$. Each clause is a subset of *literals*, which are variables and negations of variables. A clause can be viewed as the sum (or the OR) of the literals. A clause is satisfied by a truth assignment $a : \mathcal{X} \to \{\text{TRUE}, \text{FALSE}\}$ if one of the literals in the clause is TRUE.

Consider integers $k \geq 2$. A *k-CNF formula* is a CNF formula in which each clause is of size at most k. An *Ek-CNF formula* is a CNF formula in which each clause is of size exactly k.

Given a CNF formula φ, the MAX-SAT problem is to find a truth assignment satisfying the maximum number of clauses in φ. If φ is of the form X-CNF, for X $\in \{k, \text{E}k\}$, then we get the corresponding MAX-XSAT problems.

Exercise 9. Show that the problem of deciding if a 2-CNF formula is satisfiable is in P, but MAX-2SAT is NP-Hard (i.e. its decision version is **NP**-complete).

Exercise 10. State the decision version of MAX-E3SAT and show that it is **NP**-complete.

Theorem 5.1. *There exists an $8/7$-approximation algorithm for* MAX-E3SAT.

Proof. Let φ be an E3-CNF formula with m clauses $C_1, \ldots, C_m$. Let S_φ be the random variable counting the number of satisfied clauses of φ by randomly setting x_i independently to be TRUE with probability $1/2$. Since the probability that a clause C_j is satisfied is $7/8$, by linearity of expectation $\text{E}[S_\varphi] = 7m/8$. This number clearly is within a factor $7/8$ of the optimal value. Hence, this simple randomized algorithm achieves (expected) approximation ratio $8/7$. We can derandomize this algorithm by a method known as *conditional expectation*. The basic idea is as follows.

Consider a fixed $k \in [n]$. Let $a_1, \ldots, a_k \in \{\text{TRUE}, \text{FALSE}\}$ be k boolean values. Let φ' be a formula obtained by setting $x_i = a_i,\ i \leq j$, and discarding all c clauses that are already satisfied. Then, it is easy to see that

$$\text{E}[S_\varphi \mid x_i = a_i, 1 \leq i \leq k] = \text{E}[S_{\varphi'}] + c.$$

Hence, given $a_1, \ldots, a_k$ we can easily compute $\text{E}[S_\varphi \mid x_i = a_i, 1 \leq i \leq k]$ in polynomial time.

Now, for $k \geq 1$, notice that

$$\begin{aligned} & \text{E}[S_\varphi \mid x_i = a_i, 1 \leq i \leq k-1] \\ = \ & \frac{1}{2}\text{E}[S_\varphi \mid x_i = a_i, 1 \leq i \leq k-1,\ x_i = \text{TRUE}] + \frac{1}{2}\text{E}[S_\varphi \mid x_i = a_i, 1 \leq i \leq k-1,\ x_i = \text{FALSE}] \end{aligned}$$

The larger of the two expectations on the right hand side is at least $\text{E}[S_\varphi \mid x_i = a_i, 1 \leq i \leq k-1]$. Hence, we can set x_i to be TRUE or FALSE one by one, following the path that leads to the larger expectation, to eventually get a truth assignment which satisfies as many clauses as $\text{E}[S_\varphi] = 7m/8$. □

6 An FPTAS for SUBSET SUM

An instance of the SUBSET SUM problem consists of a set X of n positive integers and a target t. The objective is to find a subset of X whose sum is at most t and is as close to t as possible. In this section we give a fully polynomial time approximation scheme for this problem.

Let $X = \{x_1, \ldots, x_n\}$. Suppose we have a set S_i holding all sums of subsets of $\{x_1, \ldots, x_i\}$, then $S_{i+1} = S_i \cup (S_i + x_{i+1})$, where $S + x$ is the set of all sums of elements of S and x, for any set S and number x. This idea can be used to give an exact algorithm for the SUBSET SUM problem. Unfortunately, the algorithm runs in exponential worst case time. The lists S_i keeps growing exponentially longer, roughly doubled after each step.

The idea of our approximation scheme is to trim down S_i so that it does not grow too large. Suppose S_i is sorted in increasign order. Given a parameter $\delta > 0$, we can scan S_i from left to right. Suppose b is the current element being examined and a is the previous element, then b is removed from S_i whenever $a(1+\delta) \geq b$ (which is at least a). Basically, when δ is small enough b can be "represented" by a since they are close enough to each other. Let TRIM(S, δ) be a procedure which trims a set S with parameter δ, then TRIM can certainly be implemented in polynomial time. We are now ready to describe our FPTAS for SUBSET SUM.

FPTAS-SUBSET-SUM(X, t, ϵ)

1: $n \leftarrow |X|$
2: $S_0 \leftarrow \langle 0 \rangle$
3: **for** $i = 1$ to n **do**
4: $\quad S_i \leftarrow S_{i-1} \cup (S_{i-1} + x_i)$
5: $\quad S_i \leftarrow$ TRIM$(S_i, \epsilon/2n)$
6: $\quad$ Remove from S_i elements $> t$
7: **end for**
8: **return** a^* – the largest element in S_n

Theorem 6.1. FPTAS-SUBSET-SUM *is a fully polynomial time approximation scheme for* SUBSET SUM.

Proof. Let P_i be the set of all sums of subsets of $\{x_1, \ldots, x_i\}$. Then, it is easy to see that for any $b \in P_i$ where $b \leq t$, there is an $a \in S_i$ such that

$$b \leq \left(1 + \frac{\epsilon}{2n}\right)^i a.$$

In particular, if t^* is the optimal sum then

$$t^* \leq \left(1 + \frac{\epsilon}{2n}\right)^n a^* \leq (1+\epsilon)a^*.$$

It remains to show that the algorithm actually runs in polynomial time in n and $1/\epsilon$. Since the ith loop of the algorithm takes time polynomial in the size of S_i, it is sufficient to show that $|S_i|$ is a polynomial in n and $1/\epsilon$. Note that if $1 < a < b$ are both in S_i (after trimming), then $b/a > (1 + \epsilon/2n)$. Hence, the number of elements in S_i is at most

$$\begin{aligned} 2 + \log_{1+\epsilon/2n} t &= 2 + \frac{\ln t}{\ln(1+\epsilon/2n)} \\ &\leq 2 + \frac{2n(1+\epsilon/2n)\ln t}{\epsilon} \\ &= 2 + 2n\frac{1}{\epsilon}\ln t + 2n \ln t, \end{aligned}$$

which is clearly a polynomial in n and $1/\epsilon$. (We used the fact that $\ln x \geq x/(1+x)$.) □

7 Inapproximability

Suppose we have designed an r-approximation algorithm for some problem. How do we know that r is really the best we can do? Is there a better approximation algorithm? One way to show that r is the best approximation ratio is to show that it is **NP**-hard to approximate the problem to within any ratio smaller than r. We give several results of this favor in this section.

Theorem 7.1. *It is* $\mathbf{NP}$*-hard to approximate TSP to within any constant ratio.*

Proof. Suppose there is an r-approximation algorithm $\mathcal{A}$ for TSP where r is some constant. We shall show that there is a polynomial time algorithm that decides if a graph $G = (V, E)$ has a Hamiltonian cycle. Let $n = |V|$ and construct a complete graph $G' = (V, E')$ from G and a weight function $w : E' \to \mathbb{Z}^+$ defined by

$$w(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ rn + 1 & \text{if } (u, v) \notin E. \end{cases}$$

Run $\mathcal{A}$ on the instance G'. We claim that $\mathcal{A}$ returns a TSP tour of size at most rn if and only if G has a Hamiltonian cycle.

Note that an optimal TSP tour of G' must have size at least n. Suppose $\mathcal{A}$ returns a TSP tour of size at most rn. Then, this tour cannot use any edge $(u, v) \notin E$. Thus, this tour corresponds to a Hamiltonian cycle in G. Conversely, suppose G has a Hamiltonian cycle, then the optimal solution for G' is of cost n. Consequently, the solution $\mathcal{A}$ returns must have cost at most rn. □

Historical Notes

Texts on Linear Programming are numerous, of which I recommend [7] and [34]. For Integer Programming, [39] and [34] are suggested. Recent books on approximation algorithms include [5, 23, 32, 37]. For linear algebra, see [24, 35]. See [1, 33] for randomized algorithms, derandomization and the probabilistic methods.

The notion of an *approximation algorithm* dated back to the seminal works of Garey, Graham, and Ullman [17] and Johnson [25]. Interestingly enough, approximation algorithms were designed in the works of Graham [19], Vizing [38], and Erdös [11] before the notion of $\mathbf{NP}$-completeness came to life.

The greedy approximation algorithm for SET COVER is due to Johnson [25], Lovász [30], and Chvátal [6]. Feige [13] showed that approximating SET COVER to an asymptotically better ratio than $\ln n$ is $\mathbf{NP}$-hard.

The most popular method of solving a linear program is called the *simplex method*, whose idea is to move along edges of the feasible polyhedron from vertex to vertex. This idea dates back to Fourier (1826), and mechanized algebraically by George Dantzig in 1947 (published in 1951 [9]), who also acknowledged fruitful conversation with von Neumann. This worst-case exponential algorithm has proved to work very well for most practical problems. Even now, when we know of many other polynomial time algorithms [27, 28, 41] to solve LPs, the simplex method is still among the best when it comes to practice. The worst-case complexity of the simplex method was determined to be exponential when Klee and Minty [29] found an example where the method actually visits all vertices of the feasible polyhedron. The quest for a provably good algorithm continued until Khachian [28] devised the *ellipsoid method* in 1979. The method performs poorly in practice, however. A breakthrough was made by Karmarkar in 1984 [27], when he found a method which works in provably polynomial time, and also 50 times faster than the simplex method in his experiments. Karmarkar's method was of the *interior point* type of method,

The $8/7$-approximation algorithm for MAX-E3SAT follows the line of Yannakakis [40], who gave the first $4/3$-approximation for MAX-SAT. A 2-approximation for MAX-SAT was given in the seminal early work of Johnson [25]. Johnson's algorithm can also be interpreted as a derandomized algorithm, mostly the same as the one we presented. Later, Karloff and Zwick [26] gave an $8/7$-approximation algorithm for MAX-3SAT based on semidefinite programming. This approximation ratio is optimal as shown by Håstad [22]. The conditional expectation method was implicit in Erdős and Selfridge [12].

Until 1990, few inapproximability results were known. To prove a typical inapproximability result such as MAX-CLIQUE is not approximable to within some ratio r (unless P=NP), a natural direction is

to find a reduction from some NP-complete problem, say 3SAT, to MAX-CLIQUE which satisfies the following properties:

- given a 3CNF formula ϕ, the reduction constructs in poly-time a graph G_ϕ
- there is some poly-time computable *threshold* t such that, if ϕ is satisfiable, then G_ϕ has a clique of size at least t, and if ϕ is not satisfiable, then G_ϕ does not have any clique of size t/r or more.

If MAX-CLIQUE is r-approximable, then one can use this r-approximation algorithm, along with the reduction above, to decide if a 3CNF formula ϕ is satisfiable. The strategy is to run the algorithm on G_ϕ. If the answer is t/r or more, then ϕ is satisfiable, else ϕ is not.

Techniques for proving NP-hardness seem inadequate for this kind of *gap-producing* reductions. Intuitively, the reason is that non-deterministic Turing Machines are sensitive to small changes: the accepting computations and rejecting computations are not very far from one another (no gap). In 1990, the landmark work by Feige, Goldwasser, Lovász, Safra, and Szegedy [15] connects probabilistic proof systems and inapproximability of NP-hard problems. This has become known as **the** PCP connection. A year later, the PCP theorem - a very strong characterization of NP - was proved with the works of Arora and Safra [4], Arora, Lund, Motwani, Sudan, and Szegedy [3]. A plethora of inapproximability results using the PCP connection follow, some of them are optimal [2,10,13,16,20–22,31]. The reader is referred to recent surveys by Feige [14] and Trevisan [36] for good discussions on this point and related history.

References

[1] N. ALON AND J. H. SPENCER, *The probabilistic method*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience [John Wiley & Sons], New York, second ed., 2000. With an appendix on the life and work of Paul Erdős.

[2] S. ARORA, L. BABAI, J. STERN, AND Z. SWEEDYK, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, J. Comput. System Sci., 54 (1997), pp. 317–331. 34th Annual Symposium on Foundations of Computer Science (Palo Alto, CA, 1993).

[3] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555. Prelim. version in FOCS'92.

[4] S. ARORA AND S. SAFRA, *Probabilistic checking of proofs: a new characterization of NP*, J. ACM, 45 (1998), pp. 70–122. Prelim. version in FOCS'92.

[5] G. AUSIELLO, P. CRESCENZI, G. GAMBOSI, V. KANN, A. MARCHETTI-SPACCAMELA, AND M. PROTASI, *Complexity and approximation*, Springer-Verlag, Berlin, 1999. Combinatorial optimization problems and their approximability properties, With 1 CD-ROM (Windows and UNIX).

[6] V. CHVÁTAL, *A greedy heuristic for the set-covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.

[7] V. CHVÁTAL, *Linear programming*, A Series of Books in the Mathematical Sciences, W. H. Freeman and Company, New York, 1983.

[8] P. CRESCENZI AND V. K. (EDS.), *A compendium of NP-optimization problems*. http://www.nada.kth.se/

[9] G. B. DANTZIG, *Maximization of a linear function of variables subject to linear inequalities*, in Activity Analysis of Production and Allocation, Cowles Commission Monograph No. 13, John Wiley & Sons Inc., New York, N. Y., 1951, pp. 339–347.

[10] I. DINUR AND S. SAFRA, *On the hardness of approximating label-cover*, Inform. Process. Lett., 89 (2004), pp. 247–254.

[11] P. ERDŐS, *On even subgraphs of graphs*, Mat. Lapok, 18 (1967), pp. 283–288.

[12] P. ERDŐS AND J. L. SELFRIDGE, *On a combinatorial game*, J. Combinatorial Theory Ser. A, 14 (1973), pp. 298–301.

[13] U. FEIGE, *A threshold of* $\ln n$ *for approximating set cover (preliminary version)*, in Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996), New York, 1996, ACM, pp. 314–318.

[14] ———, *Approximation thresholds for combinatorial optimization problems*, in Proceedings of the International Congress of Mathematicians, Vol. III (Beijing, 2002), Beijing, 2002, Higher Ed. Press, pp. 649–658.

[15] U. FEIGE, S. GOLDWASSER, L. LOVÁSZ, S. SAFRA, AND M. SZEGEDY, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292. Prelim. version in FOCS'91.

[16] U. FEIGE AND J. KILIAN, *Zero knowledge and the chromatic number*, J. Comput. System Sci., 57 (1998), pp. 187–199. Complexity 96—The Eleventh Annual IEEE Conference on Computational Complexity (Philadelphia, PA).

[17] M. R. GAREY, R. L. GRAHAM, AND J. D. ULLMAN, *Worst case analysis of memory allocation algorithms*, in Proceedings of the Fourth Annual ACM Symposium on Theory of Computing (STOC), New York, 1972, ACM, pp. 143–150.

[18] M. R. GAREY AND D. S. JOHNSON, *Computers and intractability*, W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[19] R. L. GRAHAM, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[20] V. GURUSWAMI, J. HÅSTAD, AND M. SUDAN, *Hardness of approximate hypergraph coloring*, SIAM J. Comput., 31 (2002), pp. 1663–1686 (electronic).

[21] J. HÅSTAD, *Clique is hard to approximate within* $n^{1-\epsilon}$, Acta Math., 182 (1999), pp. 105–142.

[22] ———, *Some optimal inapproximability results*, in STOC '97 (El Paso, TX), ACM, New York, 1999, pp. 1–10 (electronic).

[23] D. S. HOCHBAUM, ed., *Approximation Algorithms for NP Hard Problems*, PWS Publishing Company, Boston, MA, 1997.

[24] R. A. HORN AND C. R. JOHNSON, *Matrix analysis*, Cambridge University Press, Cambridge, 1985.

[25] D. S. JOHNSON, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278. Fifth Annual ACM Symposium on the Theory of Computing (Austin, Tex., 1973).

[26] H. KARLOFF AND U. ZWICK, *A 7/8-approximation algorithm for MAX 3SAT?*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, USA, IEEE Press, 1997.

[27] N. KARMARKAR, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[28] L. G. KHACHIAN, *A polynomial algorithm for linear programming*, Dokl. Akad. Nauk SSSR, 244 (1979), pp. 1093–1096. English translation in Soviet Math. Dokl. 20, 191-194, 1979.

[29] V. KLEE AND G. J. MINTY, *How good is the simplex algorithm?*, in Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin), Academic Press, New York, 1972, pp. 159–175.

[30] L. LOVÁSZ, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[31] C. LUND AND M. YANNAKAKIS, *On the hardness of approximating minimization problems*, J. Assoc. Comput. Mach., 41 (1994), pp. 960–981.

[32] E. W. MAYR AND H. J. PRÖMEL, eds., *Lectures on proof verification and approximation algorithms*, vol. 1367 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1998. Papers from the Workshop on Proof Verification and Approximation Algorithms held at Schloß Dagstuhl, April 21–25, 1997.

[33] R. MOTWANI AND P. RAGHAVAN, *Randomized algorithms*, Cambridge University Press, Cambridge, 1995.

[34] A. SCHRIJVER, *Theory of linear and integer programming*, Wiley-Interscience Series in Discrete Mathematics, John Wiley & Sons Ltd., Chichester, 1986. A Wiley-Interscience Publication.

[35] G. STRANG, *Linear algebra and its applications*, Academic Press [Harcourt Brace Jovanovich Publishers], New York, second ed., 1980.

[36] L. TREVISAN, *Inapproximability of combinatorial optimization problems*, Tech. Rep. 65, The Electronic Colloquium in Computational Complexity, 2004.

[37] V. V. VAZIRANI, *Approximation algorithms*, Springer-Verlag, Berlin, 2001.

[38] V. G. VIZING, *On an estimate of the chromatic class of a p-graph*, Diskret. Analiz No., 3 (1964), pp. 25–30.

[39] L. A. WOLSEY, *Integer programming*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Inc., New York, 1998. A Wiley-Interscience Publication.

[40] M. YANNAKAKIS, *On the approximation of maximum satisfiability*, J. Algorithms, 17 (1994), pp. 475–502. Third Annual ACM-SIAM Symposium on Discrete Algorithms (Orlando, FL, 1992).

[41] Y. Y. YE, *Extensions of the potential reduction algorithm for linear programming*, J. Optim. Theory Appl., 72 (1992), pp. 487–498.