IIIS 2014 Spring: ATCS - Selected Topics in Optimization
Lecture date: Feb 26, 2014
Instructor: Jian Li TA: Lingxiao Huang Scribe: Zihan Tan
(Polishing not finished yet.)

1 Some Basics about Linear Programming

1.1 Basic Theorems and Terminologies (Chapter 3 in [2])

A linear programm (LP) is a mathematical optimization problem of the following form (call it *form A*):

$$\text{maximize } \langle c, x\rangle$$

$$\text{s.t. } Ax \leq b$$

where A is a matrix, x, b, c are vectors and $\langle ., .\rangle$ is the usual inner product. Sometimes, we use cx or $c \cdot x$ as a short for $\langle c, x\rangle$. $Ax \leq b$ is a set of inequalities with form $\langle a_i, x\rangle \leq b_i$ (a_i is the i^{th} row vector of A and b_i is the ith coordinate of b).

You may encounter linear programs of other forms, such as

$$\text{minimize } \langle c, x\rangle$$

$$\text{s.t. } Ax = b, \quad x \geq 0$$

Let us call the above form *form B*. As long as the objective function is linear and the constraints are linear (no matter they are equalities, inequalities, or a mixture of them). A minimization problem can be converted to a maximization problem by reversing the signs of the coefficients in the objective function. We can replace an equality constraint $a_i x = b_i$ by two inequality constraints $a_i x \geq b_i$ and $a_i x \leq b_i$. Sometimes, we want to convert an LP of form A to another equivalent LP of form B. This can be done as follows: For each inequality constraint $a_i x \leq b$ in A, add a new variable z_i. Then replace $a_i x \leq b$ by two constraints $a_i x - x = b, -x \geq 0$.

In the following, we will be working with LPs of form A.

Geometric Interpretation: Geometric intuition can be very useful in linear programming. An inequality $a_i x \leq b$ can be interpreted as follows: any feasible solution x should lie in a **halfspace** with the norm vector to be a_i. Thus, a set of inequalities require x to lie in the intersection of the set of corresponding halfspaces. Such a region is called the *feasible region* of the LP. The intersection of a set of halfspaces is a **polyhedron** or a **polytope**(if bounded) in geometric terminology. In the following, we use letter P to represent a polyhedron.

Definition 1 *A **supporting hyperplane** H of a polyhedron P is defined to be a hyperplane that has non-empty intersection with P, and all points in P that are not on the hyperplane lie exactly on the same side of H.*

Definition 2 *A **face** is defined to be the intersection of a polyhedron P and a supporting hyperplane of it.*

A polyhedron is easy to imagine in low dimensional space. However, the same task is harder in higher dimensional space, e.g., a 4-dimensional polytope (it has a bunch of 3-dimensional faces, which correspond to the usual 3-dimensional polytope we encounter in our real life. Imagine a 3-dimensional polytope, its 2-dimensional faces have very special adjacency structure. In particular, they form a planar graph. Similar things also happen in high dimensional spaces, but are more complicated). But it is still important to know the geometry structure, which gives us intuition and brings us deeper into problems.

Proposition 3 *(Face and Subsystem) Every face can be characterized by a subsystem of original linear program with all inequalities changed to be equalities. But a certain subsystem may not determine a face of polyhedron.*

From the above proposition, we can see that a face of a polyhedron is also a polyhedron (of lower dimension). A **vertex** of P can be defined as x that is determined by a full rank subsystem, i.e., the point x such that $A'x = b'$ where A' is is the combination of some rows of A and it has full rank, while b' is the vector formed by corresponding entries in b.

Linear programming has a very solid theory foundation (built upon the theory of convex geometry, in particular convex polytope). However, we only cover its basic properties in the course and its various applications in combinatorics. We refer the interested readers to [2] for more information.

Definition 4 ***Duality:*** *The dual program of the linear program (form A) is of the following form:*

$$\begin{aligned} &\textit{minimize } \langle y, b\rangle \\ &\textit{s.t. } y^T A = c \\ &y \geq 0 \end{aligned}$$

The original linear program is called the *primal*. We can derive the above dual form from the more general Lagrangian duality theory for convex programming. We will come to this point later.

Proposition 5 ***(Weak Duality)*** *For any feasible solution x for primal program and y for the corresponding dual program, we have $c^T x \leq y^T b$.*

Proposition 6 ***(Strong Duality)*** *Let x^* and y^* be the optimal solution for the primal and the dual program (if exist) respectively, then $\langle c, x^*\rangle = \langle y^*, b\rangle$.*

The proof for weak duality is straightforward. However, the proof for strong duality is somewhat complicated. In fact duality in linear program is a special case of **Lagrangian Dual** of a convex optimization problem. Thus, strong duality in linear program can be proved by **KKT condition**. Besides, **Farkas' lemma** and **simplex method** can be employed to give proof for strong duality, which can be found in any book of linear programming.

Proposition 7 *(**Complementary Slackness**)*
Let x^* and y^* be the solution of primal program and the dual program. Then,

$$y_i^*(a_i^T x^* - b_i) = 0$$

The interpretation of this proposition is the following: the value y_i^* can tell us whether the i^{th} constraint in primal program is indeed constraining the optimal solution. To be specific, $y_i^* = 0$ means that i^{th} inequality is actually not tight on the optimal solution and vice versa.

A quantitative version of the complementary slackness is the following:

$$\frac{\partial \text{OPT}}{\partial b_i} = y_i^*$$

where OPT denotes the optimal value for both the primal and dual program. From the above formula, we can also see that if $a_i x^* < b_i$ (not tight), slightly changing b_i does not affect the optimal solution since $y_i^* = 0$. The above equation is often referred to as the economical interpretation of the dual variable (or the sensitivity of the optimal solution with respect to b_i).

2 Combinatorial Optimization via Linear Programming

Linear programs are widely used in discrete combinatorial problems. We introduce some basic knowledge and fantastic examples here.

2.1 Preliminaries (Chapter 5 in [2])

Definition 8 *(**Totally Unimodular Matrix**) A matrix A is a Totally Unimodular Matrix(TUM for short) if the determinant of every square submatrix belongs to* $\{0, 1, -1\}$.

Definition 9 *(**Integral Polyhedron**) A polyhedron P is an Integral Polyhedron if every vertex of P is integral.*

Proposition 10 *If matrix A is TUM, then for every invertible square submatrix U of A, U^{-1} is integral.*

Proof: From college linear algebra, we know

$$U_{ij}^{-1} = \frac{\det(U_{ji}^*)}{\det(U)}$$

Here U^* is the adjoint matrix of U: U_{ij}^* represents the square submatrix of U without the i^{th} row and the j^{th} column. Since U is TUM, $\det(U_{ji}^*)$ is 0 or ± 1. It then follows directly that U^{-1} is integral. □

Theorem 11 *(Hoffman, Kruskal) A is TUM if and only if for any integral vector b, $P = \{x | Ax \leq b\}$ is an integral polyhedron.*

Proof: We only prove the direct part of the theorem. Since the inverse part need some mathematical construction, which is not that relevant to our topic.

Consider every vertex of P. Let x' be determined by subsystem $A'x' = b'$ where A' has full rank. Since A is TUM and A' is invertible. Thus, $x' = A'^{-1}b'$ is integral. □

2.2 Applications

2.2.1 Bipartite Matching

Consider a bipartite graph $G = (V, E)$, $V = A \cup B$, a matching is defined to be a set T of disjoint edges. Here two edges are disjoint means two edges have no vertex in common. We want to maximize the size of T. The following program characterizes the problem:

$$\begin{aligned} &\text{maximize} \sum_{e \in T} x_e \\ &\text{s.t. } \forall v, \sum_{e \in \Gamma(v)} x_e \leq 1 \\ &\forall e, x_e \in \{0, 1\} \end{aligned}$$

We assign a variable x_e to each edge, and let $\Gamma(v)$ be the set of edges adjacent to v. We call the matrix associated with the above LP the *bipartite incidence matrix*. However, it is not a linear program since we have the constraint $x_e \in \{0, 1\}$. To make it a linear program, we need to relax it as follows:

$$\begin{aligned} &\text{maximize} \sum_{e \in T} x_e \\ &\text{s.t. } \forall v, \sum_{e \in \Gamma(v)} x_e \leq 1 \\ &\forall e, x_e \in [0, 1] \end{aligned}$$

To obtain an integral optimal solution, it suffices to show that the corresponding matrix is a TUM. We need the following clean and nice math theorem:

Theorem 12 *(Ghouila-Houri) A matrix $A_{m \times n}$ is a TUM if and only if for any subset $R \subseteq [m]$, there exists a partition $R = R_1 \cup R_2$, $R_1 \cap R_2 = \emptyset$, such that $\forall j \in [n]$,*

$$\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} \in \{0, 1, -1\}$$

Proof: We only prove one direction: If A is TUM, then we can find a partition of R for any $R \subseteq [m]$. The other direction is also interesting, but less relevant to our class.

This proof essentially uses the idea of **rounding**. The high level picture of rounding is as follows: We initially have a fractional solution (which is usually very easy to obtain), and we want to round the fractional solution to an integral solution, such that the properties satisfied by the fractional solution can be (approximately satisfied by the integral solution). In our problem, we have an extremely easy fractional partition of R: R_1 and R_2 both contain $\frac{1}{2}$ fraction of every coordinate in R:

$$\sum_{i \in R_1} a_{ij} = \sum_{i \in R_2} a_{ij} = \sum_{i \in R} \frac{1}{2} a_{ij}$$

This partition satisfies the property required by the theorem. However, we are required to give an integral partition. Thus, we use the rounding technique to round a fractional solution into an integral solution, using special properties of TUM.

First, the fact that A is TUM implies that the matrix $(A, -A, I)^T$ is also a TUM. This can be immediately seen from the definition of TUM.

If p is a vector, we use $\lceil p \rceil$ (or $\lfloor p \rfloor$) to denote the vector obtained from p by taking the ceiling (or floor) operation entry-wise. Let d be the indicator vector of set R, i.e. $d_i = 1$ iff $i \in R$.

We consider the following polyhedron:

$$\{x : x^T A \leq \lceil \frac{1}{2} d^T A \rceil, x^T A \geq \lfloor \frac{1}{2} d^T A \rfloor, 0 \leq x \leq d\}$$

You should think x as an indicator variable ($x_r = 1$ means $r \in R_1$) for the partition. The right hand side of each constraint is the target we want to shoot for (basically, half of the column sum, rounded up or down).

It is easy to see that there exists a feasible solution (however fractional) $x = \frac{1}{2} d$. Since matrix $(A, -A, I)$ is TUM, by Theorem 11, the polyhedron contains an integral solution. Let the integral solution be z.

Let $R_1 = \{r \in R \mid z_r = 0\}$ and $R_1 = \{r \in R \mid z_r = 1\}$ be the partition. The rest is just to verify that this partition satisfies the theorem (do this by yourself). □

Matching in General Graph (Chapter 14 in [4]): In a general graph (with odd cycles), the above linear program is generally not integral, since the corresponding matrix is not always a TUM. However, it can be shown that the polyhedron is **half integral**, by which we mean that every vertex is a $\{0, 1/2, 1\}$ vector.

Let us go back to bipartite graphs. Consider the dual program of the bipartite matching linear program:

$$\begin{aligned} &\text{minimize} \sum_{u \in V} y_u \\ &\text{s.t. } \forall e = (u, v) \in E, y_u + y_v \leq 1 \\ &\forall u, 0 \leq y_u \leq 1 \end{aligned}$$

In fact, it is the relaxation of the following integer program:

$$\begin{aligned} &\text{minimize} \sum_{u \in V} y_u \\ &\text{s.t. } \forall e = (u, v) \in E, y_u + y_v \leq 1 \\ &\forall u, y_u \in \{0, 1\} \end{aligned}$$

This integer program characterizes another central problem in graph theory: the Vertex Cover problem. That is, we want to find a subset T of vertices of minimum size such that every edge is adjacent to at least one vertex in T. By strong duality, we know that the two program has the same optimal value. Moreover, the vertex cover LP is also integral (why? figure it out for yourself). Thus, we obtain the following theorem:

Theorem 13 *(Konig) In a bipartite graph, the size of the minimum vertex cover is equal to the size of the maximal bipartite matching.*

However, in general graphs, this theorem does not hold. In fact, Vertex Cover problem is $NP-$hard. Using linear programming technique, we can obtain a 2-approximation solution, which will be discussed in the following section.

2.2.2 Network matrix (Lecture 4 in [3])

In this section, we introduce a large class of TUM, the so called *Network Matrices*. First, let us see two special cases.

1. The **Arc-Vertex Adjacency Matrix** $M_{n\times m}$ for a directed graph is defined as follows. Let $|V| = n$ and $|E| = m$, then we first label all the vertices from 1 to n, and all the edges 1 to m respectively. For an edge $e_t = (v_i, v_j)$, let $M_{it} = 1$ and $M_{jt} = -1$, and $M_{kt} = 0$ for $k \neq i, j$.

2. Another example is **Consecutive 1 Matrix**. It is a matrix satisfying the following two properties: 1) Each entry is either 0 or 1. 2) In every row, all the 1s are consecutive. This matrix is used in many scheduling problem. For example, interval packing problem (We want to maximize the size of the set of disjoint intervals of integrals) can be solved by linear programming, while its corresponding matrix is a consecutive 1 matrix. Thus, we obtain the following corollary.

Now we introduce the definition of Network Matrix.

Definition 14 *(**Network Matrix**) For a directed tree $T = (V, E)$ and a set of ordered pairs of vertices $P \subseteq V \times V$ ($|P| = k$), with the edges of T labeled from 1 to m. The network matrix $M_{m\times k}$ is defined as follows: The rows of a network matrix correspond to arcs E. Each arc has an arbitrary orientation (it is not necessary that there exist a root vertex r such that the tree is "rooted into r" or "out of r"). Each column corresponds to an ordered pair in P. To compute the entry at row e_t and column (v_i, v_j), look at the v_i-to-v_j path p in T; then the entry is:*

1. *+1 if arc R appears forward in P,*
2. *-1 if arc R appears backwards in P,*
3. *0 if arc R does not appear in P.*

Tutte has proved the following theorem for Network Matrix.

Theorem 15 *(Tutte) A network matrix is totally unimodular.*

Now let us see a bipartite incidence matrix is a network matrices. Here is the construction. Let $G = (U, V, E)$ be a bipartite graph. Without loss of generality, assume that the edges in E are denoted by (u, v) where $u \in U$ and $v \in V$. We construct a new directed graph $G' = (V', E')$ as well as the given set of ordered pairs of nodes as follows. Let $V' = U \cup V \cup \{s\}$, and $E' = \{(u, s)\}_{u\in U} \cup \{(s, v)_{v\in V}\} \cup E$. Let each ordered pair in P relate to a directed edge (u, v). Thus, the set of ordered pairs $P = E$. It remains to verify that the the network matrix constructed above is precisely the bipartite incidence matrix.

Similarly, we can show that that an Arc-Vertex Adjacency Matrix and a Consecutive 1 Matrix are both network matrices. Therefore, by Tutte theorem, we have the following two corollaries.

Corollary 16 *Any bipartite incidence matrix is total unimodular. The LP defined in Section 2.2.1 is integral.*

Corollary 17 *Any consecutive one matrix is total unimodular. Independent Set problem can be solved in polynomial time in interval graph, by solving the corresponding LP-relaxation.*

Discussion Is the converse of Tutte's theorem also true? In other words, is a TUM always a network matrix? In general, this is not true. But Seymore has proved the following deep and amazing fact: There are essentially only two exceptions, a 7×7 matrix and a 5×5 matrix respectively (we do not write them down here). Moreover, any TUM can be constructed by some basic unimodularity-preserving operations among the network matrices, the 7×7 and 5×5 matrix.

A Side Note: P. Seymore is one of the greatest combinatorialists in modern time. He is known for the famous Graph Minor Theorem [6], the Strong Perfect Graph Theorem, and several others. His proof (collaborated with N. Robertson) of Graph Minor Theorem is published in a series of twenty journal papers spanning over 500 densely-argued pages from 1983 to 2004.

2.3 Approximation Algorithm Using Linear Programming

In complexity theory, there is a set of problems called $NP-$hard problems. So far, there is no algorithm that can solve them in polynomial time (not even in subexponential time like $2^{O(\sqrt{n})}$). Most theoretical computer scientists believe that an NP-hard problem requires exponential time $(2^O(n))$. Thus, to design polynomial time approximation algorithm for NP-hard problems is nowadays a very central topic in TCS, and the central ideology developed in this topic (polynomial time + provable approximation factor) has become gradually increasingly popular in other areas such as operation research, algorithmic game theory, databases, networking etc. Linear programming is also a very powerful and wide-used tool in this region. Next we will give some examples.

The most basic notion in the area of approximation algorithms is the *approximation factor* of an algorithm. Let OPT be the optimal solution of an optimization problem. Let SOL denote the solution given by our algorithm. To evaluate the performance of the algorithm, we care about the following factor called approximation ratio (R).

If the problem is a maximization problem, then

$$R := \inf \frac{SOL}{OPT}$$

If the problem is a minimization problem, then

$$R := \sup \frac{OPT}{SOL}$$

The supreme and the infimum are taken over all instances of the problem. The approximation factor (also called approximation ratio) is always at least one. But sometimes, for a maximization problem, we also abuse the notion and use SOL/OPT to denote the approximation ratio (which is less 1). You should not be confused.

In most of cases, especially in combinatorial optimization problems, we care multiplicative factors. But in some cases, especially in continuous optimization problems, we also use additive error factors. For example, an algorithm with additive error may have performance guarantees like $SOL \leq OPT + O(\log n)$, or $SOL \leq OPT + \epsilon$. Sometimes, such additive error may make more sense than a multiplicative error.

2.3.1 Approximation Algorithm for Vertex Cover

Let $G = (V, E)$ be the graph. A Vertex Cover problem, as stated before, can be formulated into the following integer program.

$$\begin{aligned} &\text{minimize} \sum_{v \in V} x_v \\ &\text{s.t. } x_u + x_v \geq 1 \quad \forall e = (u, v) \in E \\ &\forall v, x_v \in \{0, 1\} \end{aligned}$$

where x_v are the variables assigned to vertex v, indicating whether or not we include this vertex in set T.

Now, we show that we can easily obtain a $2-$approximation for the Vertex Cover problem.

First, we relax the vertex cover program into the following linear program by changing the domain of variables.

$$\begin{aligned} &\text{minimize} \sum_{v \in V} x_v \\ &\text{s.t. } x_u + x_v \geq 1 \quad \forall e = (u, v) \in E \\ &\forall v, x_v \in [0, 1] \end{aligned}$$

Then we use standard algorithms, such as Ellipsoid algorithm and Interior Point algorithm [1], to obtain the optimal (fractional) solution x^* to the LP relaxation.

Next we round the fractional solution x^* into an integral one $\bar{x}$ as follows.

If $x_v^* \geq \frac{1}{2}$, then $\bar{x}_v = 1$.

If $x_v^* < \frac{1}{2}$, then $\bar{x}_v = 0$.

We claim that $\bar{x}$ is a $2-$approximation solution to the Vertex Cover problem. To be more specific, we obtain the following conclusion.

Theorem 18 *(2-approx) Our algorithm outputs a $2-$approximation to the Vertex Cover problem In other words, we have*

$$\sum_{v \in V} \bar{x}_v \leq 2OPT,$$

where OPT denotes the optimal size of vertex cover.

Proof: First, since we relaxed the problem, it is immediate that

$$\sum_{v \in V} x_v^* \leq OPT.$$

Then it is also trivial from the rounding rules that

$$\sum_{v \in V} \bar{x}_v \leq 2 \sum_{v \in V} x_v^*.$$

[1] There are many versions of the simplex algorithm (depending on the pivoting rules). However, none of them is known to run in polynomial time in worse cases. So, if we use the simplex algorithm to solve the LP, it is not a polynomial time algorithm in theory. But in practice, the simplex algorithm is always a good choice for solving LP.

Combine the two results, we conclude that

$$\sum_{v \in V} \bar{x}_v \leq 2OPT.$$

□

Discussion: There are several other ways to get a 2-approximation for vertex cover. Surprisingly, the 2−approximation obtained by a simple LP rounding algorithm is generally believed to be the best possible approximation ratio we can obtain for Vertex Cover problem. There have been a sequence of papers which gives approximation algorithms with ratios $2 - o(1)$. However, recently it is shown that $2 - \epsilon$ approximation is impossible for any constant $\epsilon > 0$ if the Unique Game Conjecture (UGC) is true. UGC is a complexity hypothesis which many people believe to be true (or at least some weaker version of it). We won't tell you what exactly it is. But if you see it somewhere, just think it is like (a weaker version of) the $P \neq NP$ conjecture.

2.3.2 Approximation Algorithms for Set Cover (Chapter 14 in [4])

Another central *NP*-hard problem, known as Set Cover problem, can be approximated via linear programming as well. We give the precise statement of the problem.

Definition 19 *(Set Cover) Given a set $E = \{e_1, \cdots, e_n\}$ and a family of subsets $\mathcal{F} = \{S_1, \cdots, S_m\}$, $S_i \subset E$, we are asked to select a subset of $\mathcal{F}$ (i.e. an index set $I \subset [m]$) of the minimal size such that $E = \cup_{i \in I} S_i$.*

The Set Cover problem can be formulated as the following integer program. Let variable x_S indicate whether or not we choose set S.

$$\begin{aligned} &\text{minimize} \sum_{S \in \mathcal{F}} x_S \\ &\text{s.t. } \forall e \in E, \sum_{S: e \in S} x_S \geq 1 \\ &\forall S, x_S \in \{0, 1\} \end{aligned}$$

As before, we relax the integer programm into the following linear programming relaxation.

$$\begin{aligned} &\text{minimize} \sum_{S \in \mathcal{F}} x_S \\ &\text{s.t. } \forall e \in E, \sum_{S: e \in S} x_S \geq 1 \\ &\forall S, x_S \in [0, 1] \end{aligned}$$

Again, we use standard LP solver to obtain an optimal solution x^* for the relaxation. Then we introduce a crucial and powerful technique in rounding: **Randomized Rounding**. Randomized rounding is a class of method to round a fractional solution x^* into an integer solution $\bar{x}$ in a probabilistic fashion. For set cover, the rounding algorithm is extremely simple:

Repeat the following d times (we will fix d later) independently:

- Suppose this is the jth round. For each i, set $\bar{x}_i^{(j)} = 1$ independently with probability x_i^*, and set $\bar{x}_i = 0$ with probability $1 - x_i^*$.

Let $\bar{x}$ be our final output,

$$\bar{x}_i^{(j)} = \max_j \{\bar{x}_i$$

. If $\bar{x}_i = 1$, we choose S_i, and $\bar{x}_i = 0$ otherwise.

We claim that the randomized rounding algorithm gives us a good approximation.

Claim 20 *($\log n$-approx) Let $d = O(\log n)$, then with constant probability, the random rounding algorithm gives us an approximation algorithm with ratio $O(d)$.*

Proof: First we show that with high probability [2], the resulting solution is feasible (i.e., it covers all elements). Now, let us analyze the probability that a certain element e_t is not covered by any chosen set.

In every iteration, the probability that e_t is covered by some set is:

$$p_t = 1 - \Pi_{S:t \in S}(1 - x_t^*)$$

Since $\sum_{S:e \in S} x_S \geq 1$, let h_t denote the number of set in $\mathcal{F}$ that contains e_t the above p_t can be lower bounded by

$$p_t \geq 1 - (1 - \frac{1}{h_t})^{h_t} \geq 1 - \frac{1}{e}$$

Thus, let $d = 2 \ln n$, the probability that e_t is finally not covered is at most $(\frac{1}{e})^d \leq \frac{1}{n^2}$. By union bound, the probability that there exists any elements in E that is not covered is at most $\frac{1}{n}$. So, the probability that all elements are covered is at least $1 - 1/n$.

It remains to show the approximation ratio. According to the rounding scheme and by linearity of expectation,

$$E[SOL] = \sum_{i=1}^{m} \bar{x}_S \leq \sum_{i=1}^{m} \sum_{j=1}^{d} \bar{x}_i^{(j)} = \sum_{i=1}^{m} \sum_{j=1}^{d} x_i^* = d \cdot OPT(LP) \leq d \cdot OPT$$

where $OPT(LP)$ represents the optimal solution for the linear program relaxation, and OPT denotes the optimal solution for the Set Cover problem. By Markov inequality, we can see that $\Pr[SOL \leq 2dOPT] \geq 1/2$.

Thus, by union bound, $\Pr[SOL \leq 2dOPT$ and SOL is feasible$] \geq 1/2 - 1/n \geq 1/3$ (assuming $n > 6$). □

We can boost the success probability to "high probability" by repeating the algorithm for $O(\log n)$ times and pick the solution with minimum cost (using Chernoff bound, think about why this is true).

Greedy Algorithm: We now show a simple deterministic algorithm that gives a $\log n$ approximation algorithm for Set Cover problem. The algorithm proceeds in a greedy fashion, the statement is very simple:

[2] In this course, "with high probability" means "with probability at least $1 - 1/poly(n)$".

- In each iteration, choose S_i that covers the most number of uncovered elements.

We claim that the greedy algorithm can achieve an approximation ratio of $\ln n$.

Theorem 21 *($\log n$-approx) The greedy algorithm gives us an approximation algorithm to Set Cover problem with ratio to be $H_n = \sum_{i=1}^{n} 1/i \approx \ln n$.*

Proof: To see it is a $\log n$ approximation algorithm, we use another technique called **dual fitting** technique.

Consider the primal linear program.

$$\text{minimize} \sum_{S \in \mathcal{F}} x_S$$

$$\text{s.t. } \forall e \in E, \sum_{S: e \in S} x_S \geq 1$$

$$\forall S, x_S \in [0, 1].$$

It is dual is the following:

$$\text{maximize} \sum_{e \in E} y_e$$

$$\text{s.t. } \forall S \in \mathcal{F}, \sum_{e \in S} y_e \leq 1$$

$$\forall e, y_S \geq 0.$$

Let $p(e)$ be the price of covering element e (Suppose e is covered by S in the t^{th} iteration), defined as the inverse of the number of uncovered element covered by S in t^{th} round. In other words, if we choose S in the t^{th} iteration, and S covers c uncovered elements including e, then the price of e is $p(e) = \frac{1}{c}$.

It is immediate to see that $SOL = \sum_{e \in E} p(e)$.

Set $y_e = \frac{1}{H_n} p(e)$, where H_n is the n^{th} Harmonic number, defined as $H_n = 1 + \frac{1}{2} + \cdots \frac{1}{n}$. It suffices to prove that y_es are feasible in dual program, then by weak duality we can prove that

$$\sum_{e \in E} y_e \leq OPT(primal) \leq OPT$$

Hence, we can conclude that

$$SOL = \sum_{e \in E} p(e) = H_n \sum_{e \in E} y_e \leq H_n OPT \leq \log n OPT$$

It remains too show that y_es are feasible for the dual.

Now, we make a crucial observation for the greedy algorithm. Let $S = \{e_1, e_2, \cdots e_r\}$ be an arbitrary set in $\mathcal{F}$. Without loss of generality, assume that the order of elements being covered is $e_1, e_2, \cdots e_r$, i.e. e_j is not covered earlier than e_i if $j > i$.

Consider the price of an element e_i in S. We observe that:

$$p(e_i) \leq \frac{1}{r - i + 1}$$

This is because when e_i is chosen by the greedy algorithm, at least $r-i+1$ elements of S have not been chosen (otherwise, the greedy algorithm should choose S).

Thus, we have

$$\sum_{e \in S} y_e \leq \frac{1}{H_n} \sum_{i=1}^{r} \frac{1}{r-i+1} = \frac{H_r}{H_n} \leq 1.$$

This completes our proof. □

• **Discussion**

Surprisingly, the H_n ratio is almost tight for the set cover problem. As shown by Feige [5], $(1-\epsilon)\log n$ approximation ratio for Set Cover problem is impossible unless

$$\text{NP} \subset \text{DTIME}(2^{\log n \log\log n})$$

. Most theoretical computer scientists believe that an *NP* complete problem requires exponential time (something like $\text{DTIME}(2^{O(n)})$) to solve exactly. Even subexponential time (something like $\text{DTIME}(2^{O(\sqrt{n})})$) is too small to contain *NP*.

References

[1] Cover T M. On determining the irrationality of the mean of a random variable. Ann. Statist, 1973, 1(5): 862-871.

[2] Korte, Bernhard, Jens Vygen, B. Korte, and J. Vygen. Combinatorial optimization. Vol. 1. Berlin: Springer, 2002.

[3] Chandra Chekuri. CS 598: Topics in Combinatorial Optimization. Spring 2010. http://courses.engr.illinois.edu/cs598csc/sp2010/.

[4] Vazirani, Vijay V. Approximation algorithms. springer, 2001.

[5] Feige, U. A threshold of ln n for approximating set cover. Journal of the ACM (JACM), 45(4), 634-652, 1998.

[6] Graph Minor Theorem: *http : //en.wikipedia.org/wiki/Robertson%E2%80%93Seymour_theorem*