Chapter 1

Introduction to Approximation Algorithms

A *decision problem* is a problem Π such that every instance I of Π has a "yes/no" solution. An algorithm A that solves Π produces a correct "yes/no" answer for each instance I of Π.

Every instance I of an *optimization problem* Π has a non-empty feasible set of solutions, denoted $F_\Pi(I)$ such that associated with every feasible solution, $s \in F_\Pi(I)$, there is a non-negative rational cost, denoted $C_\Pi(I, s)$. Any feasible solution that optimizes $C_\Pi(I, s)$ is called an *optimal solution* for I, denoted $OPT_\Pi(I)$. An optimization problem Π can either be a maximization problem or a minimization problem and depending on this $OPT_\Pi(I)$ is a feasible solution that either maximizes cost or minimizes cost.

Typically, the following problems related to Π have polynomial time solutions:

- Determining if a given instance I is a legal instance of Π.
- Checking if a given solution s is feasible for a given instance I (that is, determining if $s \in F_\Pi(I)$).
- Given I and a feasible soltion s, determining the cost $C_\Pi(I, s)$.

So all of these problems are easy and the hardness of Π arises from the fact that $F_\Pi(I)$ is very large and there is no known efficient way of searching $F_\Pi(I)$ to find an optimal feasible solution. Specifically, the optimization problems we will consider will be all be NP-hard. What does it mean for an optimization problem to be NP-hard?

We can view an optimization problem Π as a decision problem by attaching to each problem instance I a rational B. So each instance of the decision version of Π is a pair (I, B). If Π is a maximization problem, then its decision version asks: *Does I have a feasible solution s with cost $C_\Pi(I, s) \geq B$?*. If Π is a minimization problem, then its decision version asks: *Does I have a feasible solution s with cost $C_\Pi(I, s) \leq B$?*. Given this, the following propositions are obvious.

Proposition 1 *If an optimization problem Π can be solved in polynomial time, then its decision version can also be solved in polynomial time.*

Proposition 2 *If the decision version of an optimization problem* Π *is NP-hard, then* Π *is also NP-hard.*

So whenever we talk about an optimization problem being NP-hard, we are actually talking about its decision version being NP-hard.

An algorithm A is a *factor-f approximation algorithm* for a minimization problem Π if

- A runs in poly-time, and
- For every instance I of Π, A finds a feasible solution s such that

$$C_{\Pi}(I,s) \leq f \cdot OPT_{\Pi}(I). \tag{1.1}$$

Note that $f \geq 1$. If Π is a maximization problem, then A is a *factor-f approximation algorithm* if A runs in polynomial-time and for every instance I of Π, finds a feasible solution s such that

$$C_{\Pi}(I,s) \geq f \cdot OPT_{\Pi}(I). \tag{1.2}$$

Note here that $f \leq 1$.

We will now discuss easy approximationm algorithms for some well-known problems. The table below shows the problems we will consider and the approximation factor f that the algorithms we present will achieve. Roughly speaking, these are the best known approximation factors for each of these problems.

Problem, Π	Factor f
Graph Coloring	$O(n^c)$ for $c < 1$
Set Cover	$O(\lg n)$
Cardinality Vertex Cover	2
Minimum Makespan	$(1+\epsilon)$
Knapsack	$(1+\epsilon)$

The approximation factor $(1+\epsilon)$ for *Minimum Makespan* and *Knapsack* problems, means that for every $\epsilon > 0$, there is an algorithm A_ϵ such that A_ϵ produces a solution that is within $(1+\epsilon)$ times the optimal. So technically speaking, here we have a family of algorithms rather than a single algorithm. This family of algorithms is called a *polynomial time approximation scheme (PTAS)*. The running time of a PTAS depends inversely on ϵ and we distinguish the case when the running time of a PTAS is a polynomial function of $1/\epsilon$. A PTAS for which this is the case is called a *fully polynomial time approximation scheme (FPTAS)*. A PTAS and an FPTAS will be defined more precisely later. We will present an FPTAS for Knapsack and a PTAS for Minimum Makespan.

Example of Approximation algorithm. A *vertex cover* for a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that for every edge $\{u, v\} \in E$, either $u \in V'$ or $v \in V'$ (or both). If G is a vertex-weighted graph with weight function $w : V \to Q^+$ then the *weight* of a vertex cover is simply the sum of the weights of the vertices in it.

Vertex Cover (VC)

Input: A vertex-weighted graph $G = (V, E)$ with weight function $w : V \to Q^+$.

Output: A vertex cover of G with minimum weight.

In the "cardinality" version of the problem, called *Cardinality Vertex Cover (CVC)*, vertices have unit weights. This essentially means that we are looking for a vertex cover with fewest vertices in it.

We want to come up with an algorithm A such that for every instance I of CVC, A produces a vertex cover s such that

$$C_{CVC}(I,s) \leq 2 \cdot OPT_{CVC}(I) \tag{1.3}$$

The problem with showing such an inequality is that we don't know anything about $OPT_{CVC}(I)$. This is the fundamental problem faced by people designing approximation algorithms Typically, to get around this problem, we first show a lower bound $LB_{\Pi}(I)$ on $OPT_{\Pi}(I)$. That is,

$$LB_{\Pi}(I) \leq OPT_{\Pi}(I) \qquad \text{for all } I \tag{1.4}$$

and then show that

$$C_{\Pi}(I,s) \leq 2 \cdot LB_{\Pi}(I) \leq 2 \cdot OPT_{\Pi}(I) \tag{1.5}$$

It turns out that it is extremely easy to obtain a lower bound on $OPT_{CVC}(I)$.

A *matching* M in a graph is a set of edges, no two of which share an endpoint. A *maximal matching* is a matching that is maximal with respect to inclusion, that is, adding any other edge to the maximal matching makes it not a matching.

Algorithm for CVC

1. Compute a maximal matching M of G.

2. Output the endpoints of the edges in M.

Lemma 3 *The above algorithm produces a vertex cover of G.*

Proof: Let V' be the set of endpoints of the edges in M. If V' is not a vertex cover, then there is an edge $\{u,v\} \in E$ such that $u \notin V'$ and $v \notin V'$. Hence, $\{u,v\}$ can be added to M and it would still be matching. This contradicts the fact that M is a maximal matching. Therefore V' is a vertex cover. $\square$

Lemma 4 *For any matching M of G and any vertex cover V' of G, $|M| \leq |V'|$.*

Proof: For every edge in M, there is at least one of its end points in V'. Since M contains edges no two of which share an endpoint, $|M| \leq |V'|$. $\square$

A corollary of the above lemma is that if OPT is the size of a minimum cardinality vertex cover of G and M is a maximal matching, $|M| \leq OPT$ If we let V' denote the output of the above algorithm, we have that $|V'| = 2 \cdot |M|$ therefore $|V'| \leq 2 \cdot OPT$. This shows that the above algorithm is a factor-2 approximation algorithm for CVC.

Remarks:

- Rather than use $OPT_{\Pi}(I)$ we will use OPT when Π and I are clear from the context. In fact, we will use OPT to denote not only the optimal cost, but also the optimal solution sometimes.

- A factor-2 approximation can also be achieved for the usual (weighted) vertex cover problem.

Chapter 2

Set Cover: Greedy Algorithm

In the last lecture we studied a factor-2 approximation algorithm for Cardinality Vertex Cover (CVC). There are three questions one can ask about that algorithm and its proof.

1. Is our analysis tight? In other words, is it the case that our algorithm, for some instance, produces a vertex cover whose size is twice the size of the optimal.

   The answer is yes. Consider the complete bipartite graph $K_{n,n}$. The algorithm produces a vertex cover of size $2n$, while $OPT = n$.

2. Is there some other algorithm that uses the same lower bound (maximal matching), but produces a factor-f approximation for some $f < 2$?

   The answer is no. Consider the complete graph K_n, for odd n. In this case, any maximal matching has size $(n-1)/2$ and $OPT = n-1$.

3. Is there some other algorithm that produces a better than 2 approximation factor?

   No one knows. This is one of the most tantalizing open questions in this area.

Set Cover. Our next example is a problem for which we will present a factor-$O(\log n)$ greedy approximation algorithm.
SET COVER (SC)
Input: Given a universe U of n elements and a collection $C = \{S_1, S_2, ..., S_k\}$ of subsets of U, and an assignment $c : C \to Q^+$ of costs to the subsets in C.
Output: A subcollection C' with the minimum cost that covers the universe.

A subcollection C' *covers* U if $\bigcup_{S \in C'} = U$. The *cost* of a subcollection is simply the sum of the costs of the subsets in it.

Example. Suppose $U = \{1,2,3,4,5,6,7,8\}$, $C = \{\{1,4\},\{2,4,7\},\{3,7,8\},\{5,6,8\},\{357\}\}$, and the costs of the sets in the collection are $3,4,2,1,7$ respectively. A subcollection C' that covers U is $\{\{1,4\},\{2,4,7\},\{5,6,8\},\{3,5,7\}\}$ and it has cost $3+4+1+7 = 15$.

There is a simple "greedy algorithm" for the set cover problem. At each step pick a subset that has the smallest cost per new element that it covers. This is also the same as saying "pick a subset that covers the most number of new elements per unit cost."

1. $A = \emptyset$

2. **while** $(U \neq \emptyset)$ **do**
3. Pick a set S_i that minimizes $\frac{\text{Cost}(S_i)}{|S_i \cup U|}$
4. $A = A \cup \{Si\}$
5. $U = U - S_i$
6. output A

How do we show that this algorithm achieves a factor-$O(\log n)$ approximation? Let $e_1, e_2, \ldots, e_n$ be the order in which elements are covered by the above algorithm, with ties broken arbitrarily. Each element $e_k \in U$ is first covered by some set S_i in Step 3. Define the *price* of e_k, denoted $price(e_k)$ as $\frac{\text{Cost}(S_i)}{|S_i \cup U|}$. We can prove the following lemma.

Lemma 5 *For each k, $1 \leq k \leq n$,*

$$price(e_k) \leq \frac{OPT}{n-k+1}.$$

Lemma 6

$$cost(A) \leq H_n \cdot OPT,$$

where H_n is the Harmonic number $1 + 1/2 + 1/3 + \cdots + 1/n$.

Proof:

$$\text{cost}(A) = \sum_{k=1}^{n} price(e_k) \leq OPT \sum_{k=1}^{n} 1/(n-k+1) = H_n \cdot OPT.$$

□

Proof: (**Lemma 1**) Let

$$C = \{S_{i_1}, S_{i_2}, S_{i_3}, \ldots, S_{i_t}\}$$

be a minimal subcollection of OPT that covers $e_k, e_{k+1}, \ldots, e_n$. Let $E = \{e_1, e_2, \ldots, e_{k-1}\}$ and let $E' = \{e_k, e_{k+1}, \ldots, e_n\}$. For $e_\ell \in E'$ let S_{i_j} be the first set that contains e_ℓ. Define

$$f(e_\ell) = \frac{\text{Cost}(S_{i_j})}{|S_{i_j} - E - S_{i_1} - S_{i_2} - \cdots - S_{i_{j-1}}|}.$$

Note that

$$\sum_{\ell=k}^{n} f(e_\ell) = \text{cost}(C) \leq OPT$$

and so there is some $e_\ell \in E'$ such that $f(e_\ell) \leq OPT/(n-k+1)$.

Just before the greedy algorithm covers e_k, none of the sets $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$ have been picked by the algorithm, because elements in E' are still uncovered. If at this stage S_{i_j} were chosen, it would assign to element e_ℓ the price

$$price(e_\ell) = \frac{\text{Cost}(S_{i_j})}{|S_{i_j} - E|} \leq f(e_\ell) \leq OPT/(n-k+1).$$

Since the greedy algorithms choses to pick a set that minimizes "price" and e_k is the next element covered, $price(e_k) \leq price(e_\ell) \leq \frac{OPT}{/}(n-k+1)$. □

Question : Is this analysis tight? Yes. Consider the following example. Let $U = \{1, 2, \ldots, n\}$. Let $C = \{S_1, S_2, \ldots, S_n, S_{n+1}\}$ be a given collection of subsets of U such that $S_i = \{i\}$ for $1 \leq i \leq n$ and $S_{n+1} = \{1, 2, \ldots, n\}$. Let $\text{cost}(S_i) = 1/i$ for $1 \leq i \leq n$ and let $\text{cost}(S_{n+1}) = 1 + \epsilon$. The greedy algorithm picks $\{S_1, S_2, \ldots, S_n\}$ for a cost of H_n, while $OPT = (1 + \epsilon)$.

Chapter 3

FPTAS for Knapsack

KNAPSACK:
Input: Objects $a_1, a_2, \ldots, a_n$ with positive integer sizes given by $size(a_i)$, $1 \leq i \leq n$, positive integer profits given by $profit(a_i)$, $1 \leq i \leq n$, and a knapsack of capacity $B \in Q^+$.
Output: A collection of objects with total size at most B and maximum total profit.

Note that the total size of a collection simply refers to the sum of sizes of the objects in the collection and similarly, the total profit of a collection refers to the sum of profits of the objects in the collection. In other words, we want to find a set $A' \subseteq \{a_1, a_2, \ldots, a_n\}$ such that $\sum_{a_i \in A'} size(a_i) \leq B$ and $\sum_{a_i \in A'} profit(a_i)$ is maximized.

Simple Dynamic Algorithm For Knapsack Let $P = max\{profit(a_i) \mid 1 \leq i \leq n\}$. Since we have n objects, nP is the maximum total profit possible. Recall that profits are positive integers. For each $p \in \{1, 2, 3, ..., nP\}$, let A_p denote the size of a smallest set with profit equal to p. Suppose we have computed A_p for every $p \in \{1, 2, ..., nP\}$ then we just have to find largest p such that $A_p \leq B$. To compute the A_p's, we first compute $A_{i,p}$ where, $1 \leq i \leq n$, $p \in \{1, 2, ..., nP\}$ and $A_{i,p}$ is defined as the size of a smallest subset of $\{a_1, a_2, \ldots, a_i\}$ with profit equal to p. Note that "size of a subset" here does not refer to the cardinality of the subset, it refers to the sum of the sizes of the objects in the set. Computing $A_{i,p}$ is essentially filling an $n \times nP$ table. Also note that $A_p = A_{n,p}$ for each p.

It is obvious that for each $p \in \{1, 2, \ldots, nP\}$,

$$A_{1,p} = \begin{cases} 0 & \text{if } p \neq profit(a_i) \\ size(a_i) & \text{if } p = profit(a_i) \end{cases}$$

For each i, $2 \leq i \leq n$, and $p \in \{1, 2, 3, ..., nP\}$

$$A_{i,p} = \begin{cases} \min\{A_{i-1,p-profit(a_i)} + size(a_i), A_{i-1,p}\} & \text{if } profit(a_i) < p \\ \min\{size(a_i), A_{i-1,p}\} & \text{if } profit(a_i) = p \\ A_{i-1,p} & \text{otherwise} \end{cases}$$

This step of computing $A_{i,p}$ for any i and p takes $O(1)$ and since we have an $n \times nP$ table to fill the total time taken is $O(n^2 P)$.

Now note that this is not a polynomial time algorithm in general. For example, assume that each of the object sizes can be represented in c bits for some constant c. Further assume that

$P = profit(a_i)$ for all i and $P = 2^n$. Then the input can be represented in $O(n^2)$ bits, while the running time of the algorithm is $O(n^2 \cdot 2^n)$. Specifically, such an algorithm is said to run in *pseudopolynomial time*. The notion of pseudopolynomial running time can be precisely defined as follows. Let I be an instance of problem and I_u denote I expressed in unary. If an algorithm for the problem runs in time polynomial in $|I_u|$ then the running time is said to be pseudopolynomial.

We now convert this dynamic programming algorithm into FPTAS (Fully Polynomial Time Approximation Scheme) for KNAPSACK. Like with other constructions of FPTASs and PTASs here we trade off accuracy for running time.

Algorithm

1. "Shrink" the profits by assigning to each a_i assign a new profit:

$$profit'(a_i) = \lfloor \frac{profit(a_i)}{K} \rfloor$$

   for some fixed positive K to be chosen later.

2. Solve KNAPSACK using the new profits and using the dynamic programming algorithm discussed earlier.

3. Report the subset produced as the solution.

Analysis

Let O be an optimal solution of KNAPSACK and let O' be solution produced by above algorithm. Clearly,

$$OPT = \sum_{a_i \in O} profit(a_i) \qquad \text{and} \qquad \sum_{a_i \in O'} profit(a_i) \leq OPT$$

Since O' is optimal in shrunken instance, we have

$$\sum_{a_i \in O'} \lfloor \frac{profit(a_i)}{K} \rfloor \geq \sum_{a_i \in O} \lfloor \frac{profit(a_i)}{K} \rfloor.$$

We can bound the lefthand side (LHS) above as follows:

$$LHS \equiv \sum_{a_i \in O'} \lfloor \frac{profit(a_i)}{K} \rfloor \leq \sum_{a_i \in O'} \frac{profit(a_i)}{K} = \frac{profit(O')}{K}.$$

Similarly, we can bound from below the righthand side (RHS) by

$$RHS \equiv \sum_{a_i \in O} \lfloor \frac{profit(a_i)}{K} \rfloor \geq \sum_{a_i \in O} (\frac{profit(a_i)}{K} - 1) \geq \sum_{a_i \in O} \frac{profit(a_i)}{K} - n = \frac{OPT}{K} - n.$$

Thus,

$$\frac{profit(O')}{K} \geq \frac{OPT}{K} - n$$

and multiplying both sides by K we get

$$profit(O') \geq OPT - Kn.$$

For a given $\epsilon > 0$, choose K such that $Kn = P\epsilon$. This implies that

$$profit(O') \geq OPT - P\epsilon$$

and without loss of generality we can assume that $P \leq OPT$. Therefore,

$$profit(O') \geq OPT - \epsilon \cdot OPT = (1 - \epsilon)OPT.$$

Running Time Analysis The dynamic programming algorithm takes $O(n^2 \lfloor \frac{P}{K} \rfloor)$ time now and since $K = \frac{P\epsilon}{n}$, the running time of the algorithm is $O(n^3/\epsilon)$. Therefore the algorithm is polynomial in n and polynomial in $1/\epsilon$.

So we have presented an FPTAS for knapsack. The precise definition of a PTAS and an FPTAS is the following. For a *minimization problem* Π, an algorithm A is a PTAS if, for every $\epsilon > 0$ and for every instance I of Π, A produces solution s such that

$$Cost_{\Pi}(I, s) \leq (1 + \epsilon)OPT.$$

The running time of the algorithm is polynomial in the size of the input, but depends arbitrarily on ϵ. A FPTAS additionally satisfies the requirement that the running time depends polynomially on $1/\epsilon$. A PTAS and an FPTAS are defined similarly for a maximization problem except that the algorithm produces a solution s such that

$$Cost_{\Pi}(I, s) \geq (1 - \epsilon)OPT.$$

Chapter 4

PTAS for Minimum Makespan

MINIMUM MAKESPAN (MMS):
INPUT: A set of n jobs with processing times $p_1, p_2, \ldots, p_n \in Q^+$ and $m \in Z^+$.
OUTPUT: An assignment of the given jobs to m machines with minimum makespan.

Consider an arbitrary assignment of the n jobs to the m machines. Let J_i be the set of jobs assigned to machine i, $1 \leq i \leq m$. The *completion time* of machine i, denoted T_i, is $T_i = \sum_{j \in J_i} p_j$. The *makespan* of the assignment is $\max_{1 \leq i \leq n} T_i$.

There are two easy lower bounds on the optimal makespan, OPT:

1. The largest processing time of any job.

$$LB1 = \max_{1 \leq j \leq n} p_j.$$

2. The average completion time of a machine.

$$LB2 = \frac{\sum_{1 \leq j \leq n} p_j}{m}.$$

It is clear that $LB1 \leq OPT$ and $LB2 \leq OPT$ and so if we let LB denote the combined lower bound, $LB = \max\{LB1, LB2\} \leq OPT$.
Here is a simple factor-2 approximation algorithm.

1. Order jobs arbitrarily.

2. Process jobs in order, assigning each job to machine with smallest completion time, currently.

Claim: Let T be the makespan of the assignment produced by the above algorithm, then $T \leq 2 \cdot OPT$.
Proof: Let machine i be the machine with maximum completion time. Let job j be the job which is assigned last to machine i. Suppose the completion time of machine i, just before job j was assigned to it is t. Then, every machine has processing time at least t, implying that $\sum_{1 \leq j \leq n} p_j \geq m \cdot t$. This implies that

$$LB \geq \frac{\sum_{1 \leq j \leq n} p_j}{m} \geq t.$$

Also, we know that $LB \geq LB1 \geq p_j$. Then we have $T = t + p_j \leq LB + LB = 2 \cdot LB \leq 2 \cdot OPT$. □

PTAS for Minimum Makespan: We want to devise a factor-$(1+\epsilon)$ approximation algorithm for MMS for any $\epsilon > 0$. In order to do this, we first establish a connection between MMS and Bin Packing.

BIN PACKING
INPUT: $t \in Q^+$ and n objects of sizes $a_1, a_2, a_3, \ldots, a_n \in (0, t]$.
OUTPUT: Minimum number of size-t bins needed to pack the objects.

Example: Let the size of the bins be t = 1. Suppose $n = 5$ and let the sizes of objects be $a_1 = 0.7$, $a_2 = 0.3$, $a_3 = 0.4$, $a_4 = 0.5$, and $a_5 = 0.4$. We can then pack $\{a_1, a_2\}$ in one bin, $\{a_3, a_4\}$ in a second bin , and a_5 by itself, in a third bin, so the number of bins used by this packing is 3.

The Bin packing problem is well-known to be NP-complete. A connection between MMS and Bin packing is as follows:

> n jobs with processing times $p_1, p_2, \ldots, p_n$ can be assigned to m machines with makespan t iff n objects with sizes $p_1, p_2, \ldots, p_n$ can be packed in m size-t bins.

Let I denote the set $\{p_1, p_2, \ldots, p_n\}$. Let $BINS(I, t)$ denote the fewest size-t bins needed to pack objects of sizes I. The connection between MMS and Bin packing implies:

$$OPT = \min\{t | BINS(I, t) \leq m\}.$$

Also note that $OPT \in [LB, 2 \cdot LB]$ and so

$$OPT = \min\{t \in [LB, 2 \cdot LB] | BINS(I, t) \leq m\}.$$

Therefore, an algorithm for MMS that does not really work is:

1. Compute LB.

2. Do binary search in the range $[LB, 2 \cdot LB]$ to find

$$\min\{t \in [LB, 2 \cdot LB] | BINS(I, t) \leq m\}.$$

This algorithm does not work for two reasons:

1. The query $BINS(I, t) \leq m$ cannot be answered in polynomial-time because the Bin packing in NP-Complete.

2. The number of iterations of the binary search is not polynomial in the input size.

To get around this problem, we connect MMS to a restricted version of Bin packing. This restricted version of Bin packing, that can be solved in polynomial time is as follows, assumes that the n object have k distinct sizes for some fixed k. Such a problem can be solved by dynamic programming in $O(n^2 k)$ time.

Suppose $\epsilon > 0$ is fixed and let $t \in [LB, 2 \cdot LB]$. We know that all objects have size at most t. Partition the range $(0, t]$ into the following intervals:

$$\Big(0, t\epsilon\Big), \Big[t\epsilon, t\epsilon(1+\epsilon)\Big), \Big[t\epsilon(1+\epsilon), t\epsilon(1+\epsilon)^2\Big), \ldots, \Big[t\epsilon(1+\epsilon)^k, t\epsilon(1+\epsilon)^{k+1}\Big),$$

where

$$t\epsilon(1+\epsilon)^k \le t < t\epsilon(1+\epsilon)^{k+1}.$$

Construst a new instance of the Bin packing problem as follows:

1. Throw away objects of size less than $t\epsilon$.
2. For each of the remaining objects, round down the size of the object to the left endpoint of the interval to which it belongs. Specifically, for an object of size p_j, find an i such that

   $$p_j \in [t\epsilon(1+\epsilon)^i, t\epsilon(1+\epsilon)^{i+1})$$

   and replace p_j by $t\epsilon(1+\epsilon)^i$ in the new instance of bin packing.

Given that there are k left endpoints in $(0, t]$, we have and instance of Bin packing with $k+1$ distinct sizes. Now k can be related to ϵ as follows. Given that k satisfies

$$t\epsilon(1+\epsilon)^k \le t < t\epsilon(1+\epsilon)^{k+1},$$

we obtain

$$k \le \log_{1+\epsilon}\left(\frac{1}{\epsilon}\right) < k+1,$$

implying that

$$k = \left\lfloor \log_{1+\epsilon}\left(\frac{1}{\epsilon}\right) \right\rfloor.$$

So, we can solve this new instance of Bin packing in time

$$O(n^{2\lfloor \log_{1+\epsilon}(\frac{1}{\epsilon})\rfloor + 2}).$$

Now we will use the solution of the new instance get a solution to the original instance.

Chapter 5

PTAS for Minimum Makespan and IP Formulations

Call each object j with $p_j \geq t\epsilon$ a *large* object and call the remaining objects, *small* objects. For each large object j, we have rounded p_j down to $p'_j = t\epsilon(1+\epsilon)^i$ for some integer $i \geq 0$ such that $p_j \in [t\epsilon(1+\epsilon)^i, t\epsilon(1+\epsilon)^{i+1})$. Now, notice that $p_j/(1+\epsilon) \leq p'_j \leq p_j$. As mentioned in the previous lecture, we can compute, in polynomial time, an optimal solution of the restricted version of Bin packing. If we expand the size of each bin in this packing from t to $t(1+\epsilon)$, then we get a packing of the large objects restored to their original sizes. To pack the small objects, each of size in the range $(0, t\epsilon)$, we use a greedy algorithm. In other words, we try to fit each small object into an existing bin, opening a new bin only when no old bins have space left for the object. So now we have a bin packing of the original instance of the problem. Let $\alpha(I,t,\epsilon)$ denote the number of bins used in this packing.

Lemma 7 $\alpha(I,t,\epsilon) \leq BINS(I,t)$

Proof: There are two cases depending on whether the greedy algorithm used to pack the small objects, used any new bins or not. First, suppose that the greedy algorithm used no new bins. This means that $\alpha(I,t,\epsilon)$ is equal to the optimal number of i bins used to pack the restricted instance of the problem. Note that in this instance we are packing only large objects and each of these objects has shrunk in size from p_j to p'_j. This implies that $\alpha(I,t,\epsilon) \leq BINS(I,t)$.

Suppose the greedy algorithm did use new bins. This means except the last opened bin, all other bins are full at least to the extent t. Hence, in *any* bin packing of the original instance with size-t bins, we must use at least $\alpha(I,t,\epsilon)$ bins. This implies that $\alpha(I,t,\epsilon) \leq BINS(I,t)$. $\square$

We would like to view $\alpha(I,t,\epsilon)$ as a quickly computable approximation for $BINS(I,t)$. Specifically, we replace the query "Is $BINS(I,t) \leq m$?" by "Is $\alpha(I,t,\epsilon) \leq m$?" The algorithm now is essentially doing a binary search in $[LB, 2LB]$ for $\min\{t \mid \alpha(I,t,\epsilon) \leq m\}$.

Since $\alpha(I,t,\epsilon) \leq BINS(I,t)$ this implies the following:

1. If query $BINS(I,t) \leq m$ has a **YES** answer, then the query $\alpha(I,t,\epsilon)$ also has a **YES** answer.

2. If query $BINS(I,t) \leq m$ has a **NO** answer, then the query $\alpha(I,t,\epsilon) \leq m$ may have a **YES** or **NO** answer. However, if $\alpha(I,t,\epsilon) \leq m$ has a **YES** answer we know that $BINS(I,t(1+\epsilon))$ has a **YES** answer.

Let $t^* = \min\{t \mid \alpha(I,t,\epsilon) \le m\}$. The above remarks imply that $t^* \le OPT \le t^*(1+\epsilon)$. Recall that $OPT = \min\{t \mid \alpha(I,t,\epsilon) \le m\}$. So, if we could find t^*, we could return $a = t^*(1+\epsilon)$ as the answer and we would have $a \le OPT(1+\epsilon)$. However, even though we can answer the query "Is $\alpha(I,t,\epsilon) \le m$" in polynomial time, we still cannot do the binary search in polynomial time. Here is how we get around this problem.

In each iteration of the search, we shrink the search interval by 1/2. The search interval is originally $[LB, 2\cdot LB]$ and so after k iterations, the search interval is of size $LB/2^k$. We stop when the interval size is at most ϵLB. This implies that

$$\frac{LB}{2^k} \le \epsilon LB < \frac{LB}{2^{k-1}}$$

and hence, $k \ge \log_2 \frac{1}{\epsilon} > k-1$. This implies that $k = \lceil \log_2 \frac{1}{\epsilon} \rceil$.

Note that the right end-point of every search interval corresponds to a **YES** answer to the query "Is $\alpha(I,t,\epsilon) \le m$?" Furthermore, for any t smaller than the left end-point of the search interval, the query has a **NO** answer. So when we stop at a search interval $[a', b']$ we know that $a' \le t^* \le b'$. So we return the right end-point b' as the result of the binary search. Since $b' - a' \le \epsilon \cdot LB$, we have that

$$b' \le t^* + \epsilon \cdot LB \le t^* + \epsilon t^* \le t^*(1+\epsilon).$$

The result b' of the binary search get multiplied by $(1+\epsilon)$ before being finally returned, and so

$$b'(1+\epsilon) \le t^*(1+\epsilon)^2 \le OPT(1+\epsilon)^2.$$

For $\epsilon < 1$, $\epsilon^2 < \epsilon$ and so for $\epsilon < 1$,

$$b'(1+\epsilon) \le OPT(1+3\epsilon).$$

Also note that for $\epsilon \ge 1$, we might as well use the simple greedy algorithm for MMS.

The running time of of this algorithm is

$$\mathbf{O}\Big(\left\lceil \log_2 \frac{1}{\epsilon} \right\rceil n^{2(\lceil \log_{1+\epsilon} \frac{1}{\epsilon} \rceil + 1)}\Big).$$

This is because there are $\lceil \log_2 \frac{1}{\epsilon} \rceil$ iterations of the binary search and in each iteration the restricted Bin packing problem is solved in $O(n^{2(\lceil \log_{1+\epsilon} \frac{1}{\epsilon} \rceil + 1)})$ time. Since the running time depends exponentially on $\frac{1}{\epsilon}$, the algorithm is a **PTAS** and not an **FPTAS**

Integer LP formulation

All the combinatorial optimization problems we have seen so far have simple integer linear programs formulations. Here are some examples.
SET COVER
Let x_i, $i = 1, \ldots, k$ be indicator variables that denote whether S_i is in the solution or not. In other words, $x_i = 1$ if S_i belongs to the solution, and $x_i = 0$ otherwise.

SET COVER consists of minimizing

$$\sum_{i=1}^{k} cost(S_i) \cdot x_i$$

subject to the constraint that each element in the universe if covered. This is equivalent to saying that for each element $j \in U$

$$\sum_{i:j\in S_i} x_i \geq 1.$$

Thus SET COVER is equivalent to the integer linear program (ILP):

$$\min \sum_{i=1}^{k} cost(S_i) \cdot x_i$$

$$\begin{aligned}
\sum_{i:j\in S_i} x_i &\geq 1 \text{ for all } j = 1, 2, \ldots, n \\
x_i &\in 0, 1 \text{ for all } i = 1, 2, \ldots, k
\end{aligned}$$

KNAPSACK

Define x_i, $i = 1, 2, \ldots, n$ as indicator variable for each object. Then the knapsack problem can be stated as the following maximization problem:

$$\max \sum_{i=1}^{n} profit(a_i) x_i$$

$$\begin{aligned}
\sum_{i=1}^{n} size(a_i) \cdot x_i &\leq B \\
x_i &\in \{0, 1\} \text{ for all } i = 1, 2, \ldots, n
\end{aligned}$$

The first constraint forces the total size of the objects chosen to be no greater than the capacity B of the knapsack.

MINIMUM MAKESPAN

Let x_{ij}, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$ be the indicator variables telling us if job i is assigned to machine j. Then

$$\sum_{i=1}^{n} x_{ij} \cdot p_i$$

is the completion time of machine j. Let T be a variable whose value is below by the completion time of all the machines. In other words,

$$T \geq \sum_{i=1}^{n} x_{ij} \cdot p_i, \text{ for all } j = 1, 2, \ldots, n.$$

Then the MINIMUM MAKESPAN problem is equivalent to:

$$\min \ T$$

$$\begin{aligned}
\sum_{j=1}^{m} x_{ij} &= 1 \text{ for all } i = 1, 2, \ldots, n \\
\sum_{i=1}^{n} x_{ij} \cdot p_i &\leq T \text{ for all } j = 1, 2, \ldots, n \\
x_{ij} &\in \{0, 1\} \text{ for all } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, m
\end{aligned}$$

The first constraint ensures that every object is assigned to exactly one machine. In each of these examples, the last constraint forces the solutions to be integral. This constraint is what makes the problems ILP, rather than just LP.

Chapter 6

Elementary LP Theory

The integer program for SET COVER is the following:
Let x_i be an indicator variable for set S_i.
Minimize

$$\sum_{i=1}^{k} x_i \cdot c(S_i)$$

subject to

$$\begin{aligned} \sum_{i:j\in S_i} x_i &\geq 1 \text{ for } j = 1, 2, \cdots, n \\ x_i &\in \{0, 1\} \text{ for } i = 1, 2, \cdots, k \end{aligned}$$

The corresponding LP-relaxation replaces $x_i \in \{0, 1\}$ by $x_i \geq 0$ for each $i = 1, 2, \cdots, k$. Recall that $x_i \leq 1$ is unnecessary.

Here is a deterministic rounding approximation algorithm for SET COVER that uses the above LP relaxation. Let f_j be the frequency of element $j = 1, 2, \cdots, n$ (that is, the number of sets S_i that j appears in). Let $f = \max_j f_j$. The algorithm provides a factor-f approximation for SET COVER.

Algorithm

1. Solve the LP-relaxation (using your favorite polynomial-time LP solver).
2. For any variable $x_i \geq \frac{1}{f}$ in the solution of the LP-relaxation computed in step 1, round x_i to 1. Round all other x_is down to 0.

Lemma 8 *The above algorithm produces a feasible solution for SET COVER.*

Proof: Note that for each element $j = 1, 2, \ldots, n$, the constraint

$$\sum_{i:j\in S_i} x_i \geq 1$$

contains f_j terms (one term for every set j belongs to). Therefore, the maximum number of terms in any such constraint is f. This implies that for each such constraint, there is a variable x_i, $j \in S_i$, such that $x_i \geq 1/f_j \geq 1/f$. This implies that x_i is rounded to 1 by the above algorithm and hence the inequality continues to be satisfied even after the rounding step, implying feasibility. □

Lemma 9 *The cost of the solution produced by the above algorithm is at most $f \cdot OPT$.*

Proof: First note that if C^* is the optimal cost of the solution to the LP-relaxation, then

$$C^* \leq OPT$$

This follows from the fact that the feasible region of the LP-relaxation contains everything that is feasible for original SET COVER IP.

Let C be the cost of the solution produced by our algorithm. Let $x = (x_1, x_2, \ldots, x_n)$ denote an optimal solution of the LP-relaxation and let $x' = (x'_1, x'_2, \ldots, x'_n)$ denote the solution after rounding. Now

$$C = \sum_{i=1}^{k} c(S_i) \cdot x'_i.$$

Also note that

$$x'_i \leq f \cdot x_i.$$

This implies that

$$C \leq f \sum_{i=1}^{k} c(S_i) \cdot x_i = f \cdot C^* \leq f \cdot OPT.$$

□

How good is this algorithm?

1. It yields a factor-2 approximation algorithm for Vertex Cover.
2. This is incomparable to the O(log n)-factor greedy approximation algorithm for Set Cover discussed earlier. (Performance varies depending on the value of f.)

LP-Based Techniques

LP-based techniques can be partitioned into two groups:

1. Algorithms that work by *rounding*:
   - Simpler, more intuitive.
   - More costly because solving an LP is relatively costly.
2. *Primal-dual schema* algorithms:
   - They are based on LP-relaxation but eventually have *combinatorial* versions.
   - Faster, because they are combinatorial.
   - More amenable to fine-tuning.

Elementary LP Theory

An LP has a linear objective function subject to linear constraints. There are various forms of writing LPs, such as standard, canonical, slack, etc.

Standard Form of LP

Minimize

$$\sum_{j=1}^{n} c_j x_j$$

subject to

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j &\leq b_i \text{ for } i = 1, 2, \cdots, n \\ x_j &\geq 0 \text{ for } j = 1, 2, \cdots, n \end{aligned}$$

All other forms of LP(maximization of objective,non-negativity and equality constraints,etc) can be easily transformed into standard form.
More compactly, given $c \in \Re^n$, $b \in \Re^m$ and $A \in \Re^{m \times n}$, LP in standard form is

$$\min c^T x$$

subject to

$$Ax \leq b, x \geq 0.$$

Note that the solution vector x belongs to $\Re^n$.

Geometric aspects of LP
The $(m+n)$ constraints define a feasible region of the LP. Each constraint corresponds to an *n-dimensional half-space*. Therefore, the feasible region is the intersection of $(m+n)$ n-dimensional half-spaces. It is well known that this is a *convex polytope* (in $\Re^n$).
If the LP has an optimal solution, then it has one at a vertex of the feasibility polytope. The LP may not have an optimal solution because either

1. Feasible region is empty
2. Feasible region is unbounded

But this is not an issue for us as we will usually be working with non-empty, bounded feasible regions.
There are three well known algorithmic techniques for solving an LP:

1. *Simplex method* (Dantzig, 1949): This is fast, but exponential in worst case.
2. *Ellipsoid method* (Khachiyan, 1979): Polynomial time, but expensive; this was an important theoretical result showing that LP was in P.
3. *Interior point methods* (Karmarkar, 1980s): Polynomial time, it competes with Simplex. Its worst case is large polynomial time.

Integrality Gap
Let Π be an optimization problem, P be an IP for it, and L be an LP-relaxation of P. Let $OPT(I)$ denote the cost of an optimal solution of Π for instance I. Let $OPT_f(I)$ denote the cost of the

optimal solution of L. For a minimization problem, $OPT_f(I) \leq OPT(I)$ for all I. The ratio

$$\begin{array}{c} sup \\ I \end{array} \frac{OPT(I)}{OPT_f(I)}$$

is the integrality gap of the (P, L) pair.

Examples

CVC: For K_3, $OPT = 2$ and $OPT_f = 1.5$

$$\Rightarrow Integrality\ Gap\ for\ CVC \geq 2/1.5$$

MMS: Consider the case of 1 job ($n = 1$) of time P and m machines , $OPT = P$ and $OPT_f = P/m$

$$\Rightarrow Integrality\ Gap\ for\ MMS \geq m,\ ie.\ unbounded$$

Situations in which good integrality gap is guaranteed: Best possible integrality gap is 1. In some cases, this is achieved. eg. Vertex cover for bipartite graphs.

Total Unimodularity. A square matrix B is *unimodular* if $det(B) \in \{+1, -1\}$. A matrix A is *totally unimodular*(TUM) if for every non-singular, square submatrix B of A, $det(B) \in \{+1, -1\}$.

Theorem 10 *Given an LP, min* $c^T x$ *subject to* $Ax \leq b$ *and* $x \geq 0$, *if* A *is TUM then every vertex of the feasibility polytope is integral, provided* b *is integral.*

Chapter 7

Total Unimodularity and Half Integrality

7.1 A Quick Recap.

Last time, we looked at situations in which LP has integral solutions. Consider an LP:

$$\min\{c^T x | Ax \leq b, x \geq 0\}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$

Theorem 11 *All vertices of the feasibility polytope are integral if A is totally unimodular (TUM) and b is integral.*

Proof: Any vertex v of the polytope is the intersection of atleast n-dimensional hyperplanes described by the L.P. constraints. In other words, there exists an $n \times n$ matrix A_s and $b_s \in \mathbb{R}^n$ such that v is described by:

$$A_s \cdot x = b_s$$

Note that some of these equations come from the non–negativity constraints. (Basically, we take $m + n$ of the inequalities and then we turn them into equalities.)

$$\begin{aligned} x &= A_s^{-1} \cdot b_s \\ &= \tfrac{1}{\det(A_s)} \cdot \mathrm{adj}(A_s) \cdot b_s \end{aligned}$$

Recall,

$$\det(A) = \sum_{j=1}^{n} a_{ij} \cdot A_{ij}, \text{ for any } i$$

where A_{ij} is a cofactor of A and A_{ij} is defined as follows:

$$A_{ij} \;=\; (-1)^{i+j} \cdot \det(M_{ij}$$

where M_{ij} is called the "minor".

$$\text{adj}(A) = \begin{pmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \dots & A_{nn} \end{pmatrix}$$

If A is TUM, then $\det(A_s) \in \{\pm 1\}$.
Also, $\text{adj}(A)$ is a matrix where $A_{ij} = \{\pm 1\}$.
It follows that if b is integral, then $\frac{1}{\det(A_s)} \cdot \text{adj}(A_s) \cdot b_s$ is an integral vector. □

7.2 Application of the Theorem

7.2.1 Maximum Matching Problem in a Bipartite Graph

Let $G = (V, E)$ be a bipartite graph. Express the maximum matching problem on G as an I.P.
Let $x_e \in \{0, 1\}$ be an indicator variable denoting the presense of edge e in the solution:

$$\max \sum_{e \in E} x_e$$

such that,

$$\sum_{e \text{ isincident on } v} x_e \leq 1, \text{ for } v \in V$$

and,

$$x_e \in \{0, 1\}, \text{ for } e \in E$$

The corresponding relaxation replaces $x_e \in \{0, 1\}$ by $x_e \geq 0$.

Claim 1 *This relaxation always has an integral solution.*

Also, consider the L.P. relaxation of the I.P. for the Vertex Cover (VC) problem.

$$\min \sum_{v \in V} c_v \cdot x_v$$

such that,

$$x_u + x_v \geq 1, \text{ for each edge } e = \{u, v\}$$

$$x_v \geq 0, \text{ for each } v \in V$$

What is the matrix corresponding to contraints other than the non-negativity constraints?
Each row of the matrix corresponds to vertices in G and each column corresponds to edges in G.
An element $A_{ve} \in \{0, 1\}$ represents whether an edge e is incident upon vertex v in G. This matrix is also known as an *incidence matrix*.

Claim 2 *The incidence matrix of a bipartite graph is* TUM.

7.2.2 Vertex Cover Problem

Likewise, consider an I.P. for the vertex cover problem of a bipartite graph. What does the matrix look like?
Each row of the matrix corresponds to edges in G and each column represents vertices of G. An element in the matrix tells us if an edge e is incident upon vertex v. This matrix is just the transpose of the incident matrix.

Theorem 12 *Let A be a matrix with entries in the set $\{-1, 0, +1\}$, such that, each column has atmost two non–zero entries. Suppose the rows of A are partitioned in two sets, namely, I_1, I_2, such that,*

1. *If a column contains two non–zero entries of the same sign then they appear in differnt partitions.*

2. *If a column contains two non–zero entries of different signs then they appear in the same partition.*

Subject to the above conditions, A is TUM.

Proof: By induction on the size of sub–matrices.
Base Case:
Claim trivially true for a 1 matrix.
Inductive Case:
Consider a $k \times k$ submatrix C. There are two cases:

- If C has a column with all zeros then $\det(C) = 0 \Rightarrow C$ is singular.

- If C has a column with exactly one non zero entry. Let this entry be in position (i, j). Then,

$$\det(C) = (-1)^{i+j} \cdot M_{ij}$$

  where M_{ij} is obtained by deleting row i and column j from C. An immediate implication then is that $\det(C) \in \{0, \pm 1\}$.

- All columns of C has two non-zero elements. This implies that the sum of all rows of C in I_1 = sum of all rows of C in I_2. Which, in turn, implies that the rows of C are **not** linearly independent, which implies that $\det(C) = 0$.

$\square$

7.3 Implication of the Theorem

A direct implication of the above theorem is that the incidence of a bipartite graph is TUM.

Corollary 13 *The incidence matrix of any directed graph is* TUM.

Proof: Denote each incoming edge with a $+1$ and outgoing edge with a -1. After that, the application of the above theorem is trivial. $\square$

7.4 Half–Integrality of the Vertex Cover Problem

In this section we present a remarkable result due to G. Nemhauser and L. Trotter.

Definition 14 *A point $x \in \mathbb{R}^n$ is* **half–integral** *if $x_i \in \{0, \frac{1}{2}, 1\}, \forall i \in \{1, 2, \dots, n\}$*

Theorem 15 *Any vertex of the feasibility polytope of the* VC *problem is half–integral.*

Proof: Assume the contrary. Hence there exists a vertex of the feasibility polytope of VC that is not half–integral. Let $x \in \mathbb{R}^n$ be that vertex. Let

$$V_+ = \{i | \frac{1}{2} < x_i < 1\}, \text{ and } V_- = \{i | 0 < x_i < \frac{1}{2}\}$$

We are assuming that $V_+ \cup V_- \neq \emptyset$ and $x = (x_1, \dots, x_n)$.
For $\epsilon > 0$, define two new points in $\mathbb{R}^n$ as follows:

$$y_i = \begin{cases} x_i - \epsilon & \text{if } i \in V_+ \\ x_i + \epsilon & \text{if } i \in V_- \\ x_i & \text{otherwise} \end{cases} \quad , \quad z_i = \begin{cases} x_i + \epsilon & \text{if } i \in V_+ \\ x_i - \epsilon & \text{if } i \in V_- \\ x_i & \text{otherwise} \end{cases}$$

Note:

$$y \neq x, z \neq x \tag{7.1}$$

$$x = \frac{1}{2}(y + z) \tag{7.2}$$

According to Claim 3 (proved subsequently), ϵ can be made small enough so that both y and z are feasible.
This implies that x is the convex combination of two points in the feasibility polytope.
The above in turn implies that x is **not** a vertex of the feasibility polytope. A contradiction.
So, x is **half–integral**. □

Claim 3 *Given x, y, z and ϵ in the above ϵ can be made small enough so that both y and z are feasible.*

Proof: Since x is a vertex of the feasibility polytope, x is feasible. This implies that,

$$x_i + x_j \geq 1, \text{ for all edges } \{i, j\}.$$

Consider all edges $\{i, j\}$, such that,

$$x_i + x_j > 1$$

and pick $\epsilon > 0$ small enough so that all such edges

$$y_i + y_j \geq 1 \;\&\; z_i + z_j \geq 1$$

Now, we consider constraints that hold tightly for x. In other words,

$$x_i + x_j = 1$$

Look at such an edge $\{i, j\}$. The only possible cases are:

$x_i = x_j = \frac{1}{2} \Rightarrow i \notin V_+ \cup V_-, j \in V_+ \cup V_- \Rightarrow y_i = z_i = x_i, y_j = z_j = x_j$
$x_i > \frac{1}{2}; x_j < \frac{1}{2} \Rightarrow i \in V_+, j \in V_- \Rightarrow y_i = x_i - \epsilon, y_j = x_j + \epsilon \Rightarrow y_i + y_j = x_i + x_j = 1$; Similarly, $z_i + z_j = 1$
Symmetric case to the above

□

Chapter 8

Randomized Rounding: Set Cover

Randomized Rounding

The main idea behind randomized rounding is the following

1 Solve the LP-relaxation in polynomial time

2 Interpret the solution obtained as a probability vector

This idea was proposed in the 1980s by Raghavan and Thompson to solve a problem in VLSI Design Automation. We present two examples of randomized rounding: Set Cover and MAX-SAT.

Set Cover

We now discuss a randomized algorithm that gives a *O(log n)* approximation algorithm for the set cover problem. Recall that the LP-relaxation for the set cover problem can be stated as follows:

$$\min \sum_{i}^{k} cost(S_i) * x_i$$

subject to the constraints

$$\begin{aligned} \sum_{i:j\epsilon S_i} x_i &\geq 1 \text{ for each } j \in U \\ x_i &\geq 0 \text{ for each } i = 1, 2, \ldots, k \end{aligned}$$

Now carry out the following two steps
Step 1: Solve the LP-relaxation and let $x = (x_1, x_2, \ldots, x_k)$ represent the solution.
Step 2: Interpret each x_i as a probability and perform the following:

for $i = 1, 2, \ldots, k$ include S_i in the solution with probability x_i

Another way of stating the above is

for $i = 1, 2, \ldots, k$ do set $y_i = 1$ with probability x_i

Here $y_i's$ represent the integral solutions for the set cover ILP.
Some remarks about the above chosen method:

- The elements in S_i are chosen independently
- The resulting solution need not be feasible

Now we proceed to analyze steps 1 and 2. Let $Y = \sum_i^k cost(S_i) * y_i$. Note that the $y_i's$ are all binary random variables and

$$\text{Prob}[y_i = 1] = x_i \qquad \text{and} \qquad \text{Prob}[y_i = 0] = 1 - x_i.$$

Therefore $E[y_i] = x_i$. Furthermore,

$$E[Y] = E[\sum_i^k cost(S_i) * y_i] = \sum_i^k cost(S_i) * E[y_i] = \sum_i^k cost(S_i) * x_i$$

The RHS represents the optimal solution of the LP-relaxation and so $E[Y] \leq OPT$. where OPT represents the optimal solution of the ILP. The probability that $j \in U$ is NOT covered is $\prod_{i:j\epsilon S_i}(1-x_i)$. Therefore the probability P_j that an element $j \in U$ is covered is given by:

$$P_j = 1 - \prod_{i:j\epsilon S_i}(1 - x_i)$$

It is easy to see that the quantity in the RHS is minimized when $x_i = 1/f_j$ where f_j is the frequency of j for all $i : j \in S_i$. Hence,

$$P_j \geq 1 - \prod_{i:j\epsilon S_i}(1 - 1/f_j) = 1 - (1 - 1/f_j)^{f_j}$$

Recall that for all real x, $e^x \geq (1 + x)$ and hence

$$e^{-1/f_j} \geq (1 - 1/f_j) \Rightarrow e^{-1} \geq (1 - 1/f_j)^{f_j}.$$

Hence,

$$P_j \geq (1 - 1/e).$$

Therefore the probability that each element is covered is at least a constant. So the expected number of elements covered is at least $(1 - 1/e)^n$. This fact motivates step 3 as follows:
Step 3: Repeat step 2 $c \log n$ times for some positive c to be fixed later.
Note that each repetition is independent of all other repetitions. Let C' be the collection of subsets thus obtained. The probability of $j \in U$ is not covered by a subset in C' is atmost $(1/e)^{clogn}$. Now pick c such that:

$$(1/e)^{clogn} \geq 1/4n \Rightarrow e^c \leq 4$$

Therefore the probability a $j \in U$ is not covered by a subset in C' is atmost $1/4n$. Let $P_j^{C'}$ represent the probability that there exists at least one $j \in U$ that is not covered.

$$P_j^{C'} \leq 1/4 \tag{8.1}$$

By using the union bound:

$$\text{Prob}[x_1 \vee x_2 \vee \cdots \vee x_n] \leq \sum_{i}^{n} \text{Prob}[x_i] \tag{8.2}$$

We get

$$E[cost(C^{'})] \leq clogn * OPT \tag{8.3}$$

Notice that (8.2) is not dependent on the fact that $x_i^{\prime}s$ are all independent. Recall Markov's inequality:

$$\text{Prob}[X \geq k] \leq \frac{E[X]}{k} \tag{8.4}$$

By using (8.3) and (8.4) we get

$$\text{Prob}[cost(C^{'}) \geq 4clogn * OPT] \leq 1/4 \tag{8.5}$$

We need two things:

- The cost($C^{'}$) should be less than 4clogn*OPT
- $C^{'}$ should be feasible

Let the probability that the above two things happen be $P_{desired}$. Then from (8.1) and (8.5) it is easy to see that:

$$P_{desired} \geq 1/2 \tag{8.6}$$

Now simply run steps 2 and 3 until the above conditions are met. By (8.6) we expect to repeat steps 2 and 3 not more than 2 times.
Note: OPT in (8.5) actually refers to the optimal solution of the ILP which is incalculable. However we could use the optimal solution of the LP-relaxation for practical purposes.

MAX-SAT

Input: A boolean formula f in CNF defined on boolean variables $x_1, x_2, \ldots, x_n$ and associated with each clause c of f a weight w_c .
Output: A truth assignment to $x_1, x_2, \ldots, x_n$ that maximizes the weight of satisfied clauses.
We now discuss a simple randomized algorithm that give a 1/2 factor approximation (We shall later use this to produce a 3/4 factor approximation scheme)
for each $i = 1, 2, \ldots, n$ do

Set $x_i = TRUE$ with probability 1/2 (independently)

Let W be the sum of the weights of satisfied edges:

$$E[W] = E[\sum_{c \epsilon C} W_c]$$

where

$$W_c = \begin{cases} 0 & \text{if c is not satisfied} \\ w_c & \text{otherwise} \end{cases}$$

$$E[W_c] = w_c(1 - 1/2^k)$$

where k represents the number of literals in clause c. Since $k \geq 1$, $(1 - 1/2^k) \geq 1/2$. Hence

$$E[W_c] \geq w_c/2$$

Therefore

$$E[W] \geq \sum_{c \in C} \frac{w_c}{2} \geq \frac{OPT}{2}$$

since

$$OPT \leq \sum_{c \in C} w_c$$

Chapter 9

Randomized Rounding: MAX-SAT

Last class we presented a factor-1/2 approximation algorithm for MAX-SAT. Now our goal is to improve this to a factor-3/4 approximation algorithm. Here is our factor-1/2 algorithm:
Algorithm 1: Set each variable x_i to TRUE independently, with probability 1/2.
Our next algorithm uses LP relaxation followed by randomized rounding.
Algorithm 2 (Randomized Rounding Algorithm)
Start with an IP for MAX_SAT. Let z_C be an indicator variable indicating if clause C is *TRUE* or *FALSE*. For each clause C, let L_C^+ denote the set of positive literals in C, and L_C^- denote the set of negative literals in C.

$$\begin{aligned}
&\max \sum_C w_C z_C \\
&\textit{subject to} \\
&z_C \leq \sum_{i \in L_C^+} x_i + \sum_{i \in L_C^-} (1 - x_i) \\
&z_C \in \{0,1\} \text{for each clause } C \\
&x_i \in \{0,1\} \text{for each } i = 1, 2, \ldots, n
\end{aligned}$$

Let $x_i = 1$ denote setting of $x_i = \mathit{TRUE}$ and $x_i = 0$ denote setting of $x_i = \mathit{FALSE}$. In the corresponding LP-relaxation, we replace
$z_C \in \{0,1\}$ by $0 \leq z_C \leq 1$, and $x_i \in \{0,1\}$ by $0 \leq x_i \leq 1$.

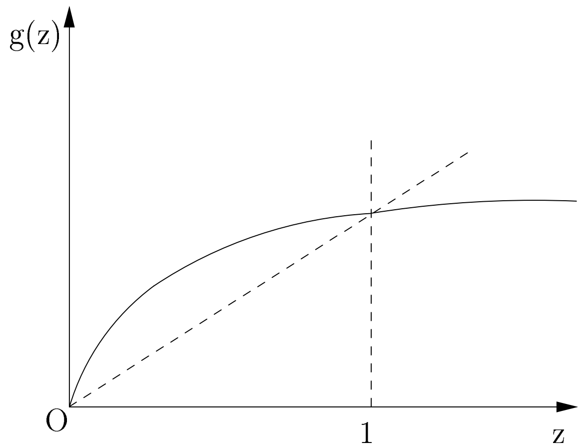
Randomized Rounding Algorithm

Step 1: Solve the LP-relaxation and let (x^*, z^*) denote an optimal solution.

Step 2: For each $i = 1, 2, 3, \ldots, n$, set $x_i = \mathit{TRUE}$ with probability x_i^*, and $x_i = \mathit{FALSE}$ with probability $(1 - x_i^*)$.

Let us analyze this algorithm. Pick an arbitrary clause C and suppose it has k literals. Without loss of generality, assume

- The literals in C involve distinct variables.
- The literals in C are all positive.



- $C = (x_1 \vee x_2 \vee x_3 \vee \cdots lor x_k)$.

Then Prob[C is FALSE] $= \prod_{i=1}^{k}(1-x_i^*)$. It is a fact that for nonnegative numbers of $a_1, a_2, \ldots, a_k$, the arithmetic mean is at least as large as the geometric mean. In other words,

$$\frac{a_1 + \cdots + a_n}{k} \geq \sqrt[k]{a_1 a_2 \cdots a_k}.$$

This implies that

$$\text{Prob}[C \text{ is FALSE}] \leq \Big(\sum_{i=1}^{k} \frac{(1-x_i^*)}{k}\Big)^k$$

Since x^* is feasible for the LP-relaxation, it satisfies

$$\sum_{i=1} k x_i^* \geq z_C^*.$$

Hence,

$$\text{Prob}[C \text{ is FALSE}] \leq (1 - \frac{z_C^*}{k})^k$$

and this implies that

$$\text{Prob}[C \text{ is TRUE}] \leq 1 - (1 - \frac{z_C^*}{k})^k.$$

We need to understand the function $g(z) \leq 1 - (1 - \frac{z}{k})^k$ better to take the next step.Suppose $\beta_k = 1 - (1 - frac1k)^k$.

Lemma 16 *$g(z) \geq \beta_k z_k$, for $z \in$[0,1].*

Proof:

$$g'(z) = -k(1 - \frac{z}{k})^{k-1}(-\frac{1}{k}) = (1 - \frac{z}{k})^{k-1}$$

$$g''(x) = (k-1)(1 - \frac{z}{k})^{k-2}(-\frac{1}{k}) < 0$$

This implies that $g'(z)$ is decreasing and the function looks as shown in the figure above. So $g(z) \geq \beta_k z$ for $z \in [0,1]$. □ This implies that Prob[C is TRUE] $\geq \beta_k z_C^*$. Therefore $E[W_C] \geq \beta_k z_C^* w_C$. We know $\beta_k = 1 - (1 - \frac{1}{k})^k \geq 1 - \frac{1}{e}$, and therefore

$$E[W_C] \geq (1 - \frac{1}{e}) z_C^* w_C.$$

This implies that

$$E[W] \geq (1 - \frac{1}{e}) \sum_C z_C^* w_C \geq (1 - \frac{1}{e}) OPT.$$

Let us reexamine the analysis of the two algorithms. Let C be a clause with k literals,
Algorithm 1: Prob[C is TRUE] $= 1 - \frac{1}{2^k} = \alpha_k$.
Algorithm 2: Prob[C is TRUE] $\geq \beta_k z_k^* = (1 - (1 - \frac{1}{k})^k) z_C^*$.

	k=1	k=2	k=3	
α_k	1/2	3/3	7/8	α_k is an increasing function of k ⇒ so algorithm 1 does well for large clauses.
β_k	1	3/3	19/27	$\beta_k$is a decreasing function of k ⇒ algorithm 2 does poorly for large clauses.

It is also easy to verify that $\alpha_k + \beta_k \geq 3/2$ for all k. This suggests a third algorithm that performs better by picking one of Algorithm 1 or Algorithm 2, randomly.

Algorithm 3: Toss a coin and run algorithm 1 or algorithm 2 depending on the outcome.

Lemma 17 $E[W] \geq \frac{3}{4} OPT$.

Proof: Let W_1 and W_2be the random variables denoting weight of solution of algorithm 1 and algorithm 2 respectively. Let C be a clause with k literals. Let W_C^1 and W_C^2 denote the random variable that stands for the weight contribution of clause C for algorithm 1 and algorithm 2 respectively. We know $E[W_C^1] = \alpha_k w_C$ and $E[W_C^2] \geq \beta_k z_C^* w_C$. Let W_C be the weight combination of clause C in combined algorithm. Then,

$$W_C = \begin{cases} W_C^1 \text{ with probability } 1/2 \\ \\ W_C^2 \text{ with probability } 1/2 \end{cases}$$

Hence,

$$E[W_C] = (W_C^1 + W_C^2)/2.$$

By substituting the bounds for the individual algorithms we get

$$E[W_C] \geq \frac{1}{2}(\alpha_k w_C + \beta_k w_C z_C^*).$$

Since $z_C^* \leq 1$, this implies

$$E[W_C] \geq \frac{1}{2}(\alpha_k w_C z_C^* + \beta_k w_C z_C^*).$$

Finally,

$$E[W_C] \geq \frac{1}{2}(\alpha_k + \beta_k) w_C z_C^* \geq \frac{3}{4} w_C z_C^*.$$

Therefore, $E[W] = \sum_C E[W_C] \geq \frac{3}{4} \sum_C w_C z_C^* \geq \frac{3}{4} OPT$. $\square$

Chapter 10

Derandomization: MAX-SAT and Scheduling on Unrelated Parallel Machines

Last week three algorithms for MAX-SAT were introduced. The final algorithm was a combination of the first two and gave us a factor-$\frac{3}{4}$ approximation algorithm. The final algorithm was:

Algorithm 3 : Toss an unbiased coin and depending on the outcome, pick algorithm 1 or algorithm 2 and run it.

For this algorithm we showed that if W is the random variable denoting the weight of the satisfied clauses then $E[W] \geq \frac{3}{4} \cdot OPT$. Alternately, a factor-$\frac{3}{4}$ approximation algorithm for MAX-SAT is given as:

Run algorithm 1 and algorithm 2 respectively, and pick the better solution.

Claim: Let W' denote the random variable that is the weight of clauses satisfied by the alternate algorithm, then $E[W'] \geq \frac{3}{4} \cdot OPT$. **Proof:**

$$W' = \max\{W^1, W^2\} \geq \frac{W^1 + W^2}{2}$$

obtain expectation of both sides,

$$E[W'] \geq \frac{E[W^1] + E[W^2]}{2}$$

the right side is the expected weight of solution if we use algorithm 3, which $\geq \frac{3}{4} \cdot OPT$ □

Derandomization. For each of these algorithms, we have only given approximation guarantees in an expected sense, which means it does not *guarantee* that we will not get a bad solution. However, both Algorithms 1 and 2 can be derandomized, that is, we can construct equivalent deterministic algorithms. We will derandomize Algorithm 1 using the technique of *conditional probabilities* to have a guaranteed factor-$\frac{1}{2}$ approximation.

In the computation tree for MAX-SAT, a level k node is identified by the k-tuple of the truth values $(a_1, a_2, a_3, \ldots, a_k)$, where $x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k$ So each leaf of the computation tree represnets a truth assignment Define the conditional expectation of a node $(a_1, a_2, a_3, \ldots, a_k)$ as

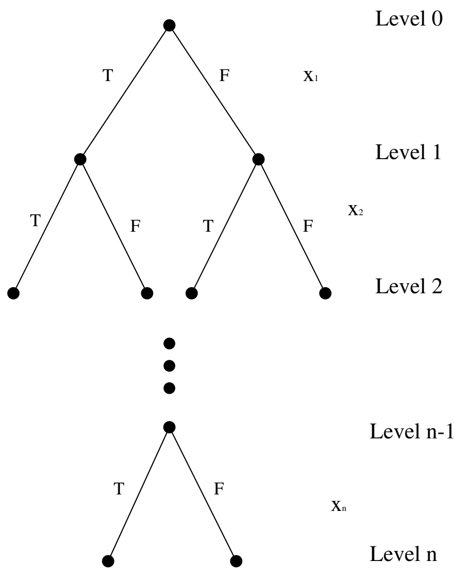


Figure 10.1: Computation tree for MAX-SAT

$E[W|x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k]$

From the definition and previous algorithms, we obtain:

Remarks:

- The conditional expectation of the root is $E[W]$.
- The conditional expectation of a leaf $(a_1, a_2, a_3, \ldots, a_n)$ is the sum of the weights of satisfied clauses obtained by setting $x_i = a_i, i = 1, 2, \ldots, n$
- The conditional expectation of any node can be computed in polynomial time.

Lemma 18 *We can compute a path from root to a leaf, such that the conditional expectation at every node in the path* $\geq E[W]$.

Proof: A node $(a_1, a_2, a_3, \ldots, a_k)$ has two children $(a_1, a_2, a_3, \ldots, a_k, T)$, and $(a_1, a_2, a_3, \ldots, a_k, F)$. Since we toss a coin to decide the truth value for x_{k+1}, the conditional expectation at this node is

$$\begin{aligned} E[w|x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k] &= \frac{1}{2}E[w|x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k, x_{k+1} = T] \\ &+ \frac{1}{2}E[w|x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k, x_{k+1} = F] \end{aligned}$$

If at node $(a_1, a_2, a_3, \ldots, a_k)$

$$E[w|x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k] \geq E[W],$$

then the conditional expectation of at least one child $\geq E[W]$. This implies that if we go to the child with a higher conditional expectation until we reach a leaf, the conditional expectation at every node in the path $\geq E[W]$. $\square$

So at the end of this path, the truth assignment represented by the leaf gives a solution which satisfies $W \geq E[W^1] \geq \frac{1}{2} \cdot OPT$

For the Algorithm 2 also, though conditional expectation at a note might not be the average of conditional expectations at its children, it is still true that the conditional expectation of at least one child $\geq E[W]$. So this works for Algorithm 2 as well.

Scheduling on unrelated parrellel machines (SUPM)
INPUT: A set J of jobs and a set M of machines, for each job $j \in J$ and machine $i \in M$, a processing time $p_{ij} \in Z^+$.
OUTPUT: An assignment of jobs to machines with minimum makespan.

Current status of this problem:

- A factor-2 approximate algorithm using LP-relaxation.
- A factor-1.5 hardness approximate algorithm (that is, if there exsits a factor-1.5 approximation algorithm, then $P = NP$).

Here is the IP for SUPM

$$\min t$$

such that

$$\begin{aligned} t &\geq \sum_{j \in J} p_{ij} \cdot x_{ij} \text{ for each } i \in M \\ \sum_{i \in M} x_{ij} &= 1 \text{ for each } j \in J \\ x_{ij} &\in \{0,1\}\ \forall i \in M, j \in J \end{aligned}$$

And the LP-relaxation for SUPM is obtained by replacing $x_{ij} \in \{0,1\}$ by $x_{ij} \geq 0,\ \forall i \in M, j \in J$

The integrality gap between this IP and LP-relaxation is huge! Let OPT be the optimal makespan, and OPT_f be the optimal makespan of the LP-relaxation. Consider such a case where there is a single job, so

$$J = \{1\}, \quad M = \{1, 2, \ldots, m\}$$

$$p_{ij} = T \text{ for all } i \in M$$

and

$$OPT = T, \quad OPT_f = \frac{T}{m}(i.e.,\ x_{i1} = \frac{1}{m} \text{ for } i \in M)$$

So the gap is m, and one might wonder if we can add constraints to the LP-relaxation to reduce this gap. Let (x, T) be a feasible solution of the IP. If $p_{ij} > T$ then job j is not assigned to machine i by the IP, and so $x_{ij} = 0$. However X_{ij} may be non-zero for the LP-relaxation. We could add a constraint C to the LP-relaxation as follows:

Constraint C: For each $i \in M, j \in J$, if $p_{ij} > T,$ then $x_{ij} = 0$

The problem here is that this is not a linear constraint. Let T^* be solution of the LP-relaxation with constraint C. Then

$$OPT_f \leq T^* \leq OPT$$

We know that the integrality gap between OPT and OPT_f is large, but the gap between OPT and T^* could be small. In fact, this is the case. So our algorithm for the problem will be:

Algorithm:

Step1. Compute (x^*, T^*) using parametric pruning.

Step2. Use rounding to go from x^* to an integral solution with makespan $T \leq 2 \cdot T^* \leq 2 \cdot OPT$.

Chapter 11

Deterministic Rounding: Scheduling on Unrelated Parallel Machines

Recall from last class that the integer program for Scheduling on Unrelated Parallel Machines (SUPM) is

$$\min t$$

subject to

$$\begin{aligned} t &\geq \sum_{j \in J} x_{ij} \text{ for each machine } i \in M \\ \sum_{i \in M} x_{ij} &= 1 \text{ for each job } j \in J \\ x_{ij} &\in \{0,1\} \text{ for all } i,j \end{aligned}$$

Recall that the difference between MINIMUM MAKESPAN and SUPM is that in SUPM the processing time of each task is machine dependant. The LP relaxation is obtained by replacing the constraint $x_{ij} \in \{0,1\}$ by $x_{ij} \geq 0$ for all $i \in M, j \in J$. We had two observations from last class:

1. The integrality gap of the above IP, LP relaxation pair is $\geq M$.
2. In any feasible solution (x,t) of the IP, if $p_{ij} > t$ then $x_{ij} = 0$.

However, item (2) above need not to be satisfied by a feasible solution of the LP. So we would like to add this constraint to the LP.

(C) if $p_{ij} > t$ then $x_{ij} = 0$ for each $i \in M, j \in J$

We will call the problem obtained by adding (C) to the LP relaxation, LP + (C). Let T^* be the makespan of an optimization solution of LP + (C), then

$$OPT_f \leq T^* \leq OPT.$$

Here, OPT_f is the cost of an optimal solution of the LP relaxation. So T^* is a better lower bound on OPT. But (C) is not a linear constraint. So to obtain T^* we use a technique called *parametic pruning*.

For any T, define $S_T = \{(i,j) \mid i \in M, j \in J, p_{ij} \leq T\}$. If (x,T) is a feasible solution of LP + (C) then $x_{ij} = 0$ for all $(i,j) \notin S_T$. This means that (x,T) satisfies the following constraints:

$$\sum_{j:(i,j)\in S_T} x_{ij} \cdot p_{ij} \leq T \text{ for all } i \in M \tag{11.1}$$

$$\sum_{i:(i,j)\in S_T} x_{ij} = 1 \text{ for all } j \in J \tag{11.2}$$

$$x_{ij} \geq 0 \text{ for all } (i,j) \in S_T \tag{11.3}$$

Conversely, if x is a feasible solution of the above set of constraints, then (x,T) is a feasible solution to LP + (C).

Now, how do we compute T^*? Suppose we know that T^* is in some range $[LB, UB]$. We simply need to find a smallest $T \in [LB, UB]$ such (x,T) is a feasible solution to set of constraints. We do this by binary search. Specifically, we start with an initial T being the midpoint of the range $[LB, UB]$. We then check of constraints (1-3) defined with respect to T has a feasible solution x. Without loss of generality, we can assume that if constraints (1-3) have a feasible solution x, then x is an extreme point of the set of feasible solutions. If constraints (1-3) do have a feasible solution x, we try a smaller value of T, else we try a larger value of T.

In this manner we compute a pair (x^*, T^*) an optimal solution to LP + (C) and we assume. Note that x^* satisfies the property that $x^*_{ij} = 0$ for all $i \in M, j \in J : p_{ij} > T^*$. It is possible that for some pairs (i,j), x^*_{ij} may be fractional and we show how to round these.

Let $r = |S_T|$. This means that x^* is a vertex of an r-dimensional polytope. Hence x^* is the intersection of some r hyperplanes, from among those defined by

$$\sum_{j:(i,j)\in S_T} x_{ij} P_{ij} = T^* \text{ for all } i \in M \tag{11.4}$$

$$\sum_{i:(i,j)\in S_{T^*}} x_{ij} =i \quad 1 \text{ for all } j \in J \tag{11.5}$$

$$x_{ij} = 0 \text{ for all } (i,j) \in S_{T^*} \tag{11.6}$$

At most $(n+m)$ of these came from the first two sets of hyperplanes. Hence $r-(n+m)$ of them have to be from the set (6). These $r-(n+m)$ constraints make $r-(n+m)$ of the r x_{ij} variables 0. These means that at most $(n+m)$ of the x_{ij} variables can be non-zero.

Let a job $j \in J$ be integral if $x^*_{ij} = 1$ for some $i \in M$. Let a job $j \in J$ be fractional if $x^*_{ij} \in (0,1)$ for some $i \in M$. How many fractional jobs can there be? Let a be the number of integral jobs and b be the number of fractional jobs. So we have that $a + b = n$. We also know that in addition to the a x_{ij} variables that are set to 1 because of the a integral jobs, there are at least $2b$ x_{ij} variables that are set to positive fractional values. Hence, we also have $a + 2b \leq (n+m)$. Solving for a and b, we get that $b \leq m$. In other words, there are at most m fractional jobs. If we could take each fractional job j and assign of it to a distinct machine i with $x^*_{ij} > 0$, we are done. This is because for any $x^*_{ij} \in (0,1)$, we have $p_{ij} \leq T^*$. Therefore, if we could find a matching of jobs to machines, our makespan increases at most by T^*. Before these fractional jobs are assigned, the makespan is T^*. After the fractional jobs are assigned, the makespan $\leq 2T^* \leq 2OPT$.

Chapter 12

Capacitated Vertex Cover

We will quickly wrap up our discussion of the problem Scheduling on Unrelated Parallel Machines (SUPM) that started last lecture. We now have a solution $(T^\star, X^\star)$ to LP + (C). We showed that at most m jobs are assigned fractionally in $x^\star$. There is a simple way in which these fractional jobs can be rounded.

Consider the bipartite graph $G = (A, B, E)$ such that A is the set of machines which are assigned some fractional jobs, B is the set of jobs assigned fractionally, and E contains edges $\{i, j\}$, where $i \in A$, $j \in B$ and $x_{ij} \in (0, 1)$. This is a bipartite graph and it can be shown that this contains a matching in which all jobs are matched. This is left as an exercise for you.

As mentioned in the last lecture, in order to "round" $x^\star$ given the above property, we simply assign each job to the machine it is matched with. This increases the makespan from $T^\star$ to at most $2 \cdot T^\star \leq 2 \cdot OPT$.

Family of Tight Examples. Let $n = m^2 - m + 1$ where n is the number of jobs and m denote the number of machines. Suppose job-1, j_1, has a processing time m on all machines and any other job, j_i, can be processed in unit time on any machine.

The OPT for this problem instance is m. Say, j_1 is assigned to m_1 and completes in time m. The remaining $m^2 - m$ jobs are assigned so that each of the remaining $m - 1$ machines get m jobs. In fact, the above solution is a feasible solution for the LP. Let us consider the following feasible solution.

- Split j_1 into unit sized jobs and assign one unit to each machine.
- Of the remaining jobs, assign $(m - 1)$ of these to each machine.

This solution is a vertex of the feasibility polytope and forms a feasible solution with makespan m. If this solution is returned by the LP relaxation, then rounding will assign j_1 to one of the machines and increase the makespan to $2m - 1$.

CAPACITATED VERTEX COVER (CapVC)
INPUT: Let $G = (V, E)$ is a graph with vertex weights $w_v \in Q^+$ and vertex capacities $k_v \in Z^+$.
OUTPUT: A vertex cover defined by a function, $x : V \to N_0$ such that

(i) There is an orientation of the edges such that the number of edges coming into any vertex is atmost $k_v \cdot x(v)$.

(ii) $\sum_{v \in V} w_v \cdot x(v)$ is minimized.

Status of the Problem: A factor-2 approximation can be obtained by using dependent rounding and an alternate factor-2 approximation algorithm can be obtained using the primal–dual framework. We will discuss a simple factor-4 algorithm that uses a deterministic rounding technique and a factor-3 approximation algorithm using dependent rounding method. Given below is the integer program corresponding to CapVC. The variables used are: $x_v \in N_0$ for each $v \in V$ and $y_{e,v} \in \{0,1\}$ for each edge $e \in E$ and $v \in e$. $y_{e,v}$ indicates if vertex v covers e.

$$\text{minimize} \sum_{v \in V} w_v \cdot x_v$$

such that

$$\begin{aligned} y_{e,v} + y_{e,u} &\geq 1 \text{ for each edge } \ e = \{u, v\} \in E \\ \sum_{e:v \in e} y_{e,v} &\leq k_v \cdot x_v \text{ for each } \ v \in V \\ x_v &\in N_0 \\ y_{e,v} &\in \{0,1\} \end{aligned}$$

The corresponding LP relaxation replaces the constraints $y_{e,v} \in \{0,1\}$ by $y_{e,v} \geq 0$ and $x_v \in N_0$ by $x_v \geq 0$. Any feasible solution to the above IP satisfies the following property:

If $y_{e,v} = 1$ for some edge $e : v \in e$, then $x_v \geq 1$.

This property can be enforced in the LP relaxation problem by adding the following linear constraint

$$x_v \geq y_{e,v} \quad \text{for each} \quad e : v \in e$$

Deterministic Rounding Algorithm. Here is a deterministic rounding algorithm that yields a factor-4 approximation.

1. Solve the LP-relaxation to obtain the solution (X, Y).
2. For each $y_{e,v} \geq \frac{1}{2}$, $\ y^\star_{e,v} = 1$. For all other $y_{e,v}$, $\ $ set $y^\star_{e,v} = 0$.
3. Set
$$x^\star_v = \lceil \frac{\sum_{e:v \in e} y^\star_{e,v}}{k_v} \rceil \tag{12.1}$$

<u>Claim</u>: This algorithm produces a factor–4 approximation algorithm.

Proof: We know that $y^\star_{e,v} \leq 2y_{e,v}$ $\forall e \in E,\ v \in e$.
And we want to show that: $\quad x^\star_v \leq 4x_v \quad \forall v \in V$.
Since,

$$y^\star_{e,v} \leq 2 \cdot y_{e,v}$$

we get,

$$y_v^\star = \sum_{e:v\in e} y_{e,v}^\star \ \le \ 2\cdot \sum_{e:v\in e} y_{e,v} \ \le \ 2\cdot k_v\cdot x_v \tag{12.2}$$

Let,

$$y_v^\star = ak_v + b \quad \forall\ a,b \in I, a\ge 0, \quad 0\le b\le k_v. \tag{12.3}$$

So,

$$x_v \ge \frac{ak_v+b}{2k_v} = \frac{a}{2} + \frac{b}{2k_v} \tag{12.4}$$

Now using (1) and (3),

$$x_v^\star = \lceil \frac{y_v^\star}{k_v} \rceil \quad \le \quad a+1 \tag{12.5}$$

Therefore, if we can show that

$$(a+1) \le 4(\frac{a}{2} + \frac{b}{2k_v}) \le 2a + \frac{2b}{k_v}$$

we will be done.

Now, $RHS = 2a + \frac{2b}{k_v}$ If $a \ge 1$ then $RHS \ge LHS$. If $a = 0$, then $y_v^\star \le k_v$. This implies that $x_v^\star \in \{0,1\}$. If $x_v^\star = 0$, then we are done. If $x_v^\star = 1$, then $y_v^\star = 1$ Hence, $y_{e,v}^\star = 1$ for some edge $e : v \in e$. This implies $y_{e,v} \ge \frac{1}{2}$ for some $e : v \in e$. Therefore, $x_v \ge \frac{1}{2}$ by the constraint added to the LP relaxation. Hence, $x_v^\star \le 4x_v$. $\square$

Chapter 13

Dependent Rounding for Capacitated Vertex Cover

Capacitated Vertex Cover

A factor-3 approximation algorithm for capacitated vertex cover
To prove this we use a very nice technique called Dependent Rounding

Dependent Rounding :
Given a bipartite graph $G = (A, B, E)$ such that each edge $\{i, j\} \in E$ has an associated real number $x_{ij} \in [0, 1]$

Goal:
To devise an efficient procedure that rounds each x_{ij} probabilistically to $X_{ij} \in \{0, 1\}$ such that the following properties are satisfied

(P1) $\text{Prob}[X_{ij} = 1] = x_{ij}$

(P2) Let

$$d_i = \sum_{j:\{i,j\} \in E} x_{ij}$$

this is called the fractional degree of i
Let

$$D_i = \sum_{j:\{i,j\} \in E} X_{ij}$$

this is called the integral degree of i
Key Property:

$\text{Prob}[D_i \in \{\lfloor d_i \rfloor, \lceil d_i \rceil\}] = 1$
If d_i is integral then $D_i = d_i$ with certainty

(P3)Negative correlation property

Cosider the following:
Assign $X_{ij} = 1$ with probabiliy x_{ij} and $X_{ij} = 0$ with probabiliy $1 - x_{ij}$ independently
Clearly (P1) is satisfied , however (P2) is not

Why is dependence useful:
Cosider randomized rounding for set cover, we are not guaranteed to get a feasible solution, that is the motivation for the property (P2) in which we do dependent rounding to get a feasible solution (P2) works only for bipartite graphs.

The Dependent Rounding scheme:

Let $x_{ij} \in [0, 1]$ be a variable associated with edge $\{i, j\} \in E$
Initially $y_{ij} = x_{ij}$ for each $(i, j) \in E$. We probabilistically modify y_{ij} in atmost $|E|$ steps such that $y_{ij} \in \{0, 1\}$ at the end.
Then we set $X_{ij} = y_{ij}$ for all $(i, j) \in E$
Our iterations will satisfy the following two invariants:

(I1) For all $(i, j) \in E, y_{ij} \in [0, 1]$.

(I2) Call $(i, j) \in E$ rounded if $y_{ij} \in \{0, 1\}$, and floating if $y_{ij} \in (0, 1)$.Once an edge gets rounded, y_{ij} never changes.

An iteration proceeds as follows. Let $F \subseteq E$ be the current set of floating edges. If $F = \phi$, we are done.Otherwise, find(in linear time) a simple cycle or maximal path P in subgraph (A, B, F), and partition the edge set of P into two matchings M_1 and M_2.Note that such partitions exists since (A,B,E) is a bipartite graph.
If P is a cycle then it is an even cycle

Define
$\alpha =$ maximum mass we can transfer from M_1 to M_2.

$$\alpha = min\{x_{ij} | \{i, j\} \in M_1\} \cup \{1 - x_{ij} | \{i, j\} \in M_2\}$$

$\beta =$ maximum mass we can transfer from M_2 to M_1.

$$\beta = min\{x_{ij} | \{i, j\} \in M_2\} \cup \{1 - x_{ij} | \{i, j\} \in M_1\}$$

We can transfer α from M_1 to M_2 with probability $\frac{\alpha}{\alpha+\beta}$ or
β from M_2 to M_1 with probability $\frac{\beta}{\alpha+\beta}$

Capacitated vertex cover:

$$min \sum_{v \in V} w_v x_v$$

such that :

$$y_{e,v} + y_{e,u} \geq 1 \;\; \forall e = (u,v) \in E$$

$$\sum_{e:v \in e} y_{e,v} \leq k_v x_v \;\; \forall v \in V$$

$$x_v \geq y_{e,v} \;\; \forall v, \forall e : v \in e$$

$$x_v \geq 0 \; \forall v \in V$$

$$y_{e,v} \geq 0 \;\; \forall e \in E, v \in e$$

Threshold and Round:

Step1: Solve the LP-Relaxation , Let solution be (X, Y)
Step2(Threshold): For each edge $e = \{u, v\} \in E$, if $y_{e,u} \geq \frac{2}{3}$ set $y^*_{e,u}$ to 1 else if $y_{e,v} \geq \frac{2}{3}$ set $y^*_{e,v}$ to 1.
Step3(Rpunding Step):
Let E' be the set of edges e for which $y_{e,u} \leq \frac{2}{3}$ and $y_{e,v} \leq \frac{2}{3}$
Create a bipartite graph $H = (E', V, E'')$ where V = vertices of the original given graph.
E' connects each edge E in E' to its end points in V.Constraint (P2) is satisfied

Run dependent rounding on H to get $y^*_{e,u}$ valus for the remaining edges $e \in \{u, v\}$
We finally set

$$x^*_v = \lceil \frac{\sum_{e:v \in e} y^*_{e,v}}{k_v} \rceil$$

$\forall v \in V$

Claim:

$$E[X^*_v] \leq 3x_v \; \forall v \in V$$

$$E[\sum_{v\in V} w_v x_v^*] \leq 3\sum_{v\in V} w_v x_v \leq 3OPT$$

Proof:

Let

$$y_v^* = \sum_{e:v\in e} y_{e,v}^*$$

$$x_v^* = \lceil \frac{y_v^*}{k_v} \rceil$$

$$y_v^* = d_v^* + r_v^*$$

where d_v^* is the number of edges assigned to V in the threshold step and r_v^* is the number of edges assigned to V in dependent rounding step

r_v^* is the integral degree of V

$$r_v^* \in \{\lfloor r_v \rfloor, \lceil r_v \rceil\}$$

where r_v is the fractional degre of v.
so,

$$r_v = \sum_{e:v\in e, e\in E'} y_{e,v}$$

Suppose

$$r_v = a_v + f_v$$

where a_v is the integral part and f_v is the fractional part

$$r_v^* \in \{a_v, a_v + 1\}$$

We can say that,

$$Prob[r_v^* = a_v] = 1 - f_v$$

$$Prob[r_v^* = a_v + 1] = f_v$$

because that is how we performed the dependent rounding

$$E[x_v^*] = (1 - f_v)\lceil\frac{d_v^* + a_v}{k_v}\rceil + (f_v)\lceil\frac{d_v^* + a_v + 1}{k_v}\rceil$$

Proof:
Two cases:
CASE1:

$$d_v^* + a_v < k_v$$

$$d_v^* + a_v + 1 < k_v$$

implies that $E[X_v^*] \leq 1$
We only have to show that $x_v \geq \frac{1}{3}$

If $d_v \geq 1$, then $y_{e,v} \geq \frac{2}{3}$ for some edge e incident on V.
so, $y_{e,v} \geq \frac{2}{3}$ and $x_v \geq \frac{2}{3}$
If $d_v = 0$ and there is some edge in $e \in E'$ incident on V then

$$y_{e,v} > \frac{1}{3}\ ,\ x_v > \frac{1}{3}$$

If $d_v = 0$ and there is some edge $e \in E'$ incident on V then all the edges were already assigned to other end points which implies that $x_v = 0$

CASE2: $d_v^* + a_v \geq k_v$

$$E[X_v^*] \leq (1 - f_v)(\frac{d_v^* + a_v + k_v}{k_v}) + f_v(\frac{d_v^* + a_v + 1 + k_v}{k_v})$$

$$\leq \frac{d_v^* + a_v + k_v}{k_v} + \frac{f_v}{k_v} \leq 2(\frac{d_v^* + k_v}{a_v}) + \frac{f_v}{k_v} \leq \frac{2(d_v^* + a_v + f_v)}{k_v}$$

Claim:

It can be written as

$$3[\frac{\frac{2}{3}(d_v^* + a_v + f_v)}{k_v}] \leq 3x_v$$

Proof

We know that the sum $a_v + f_v$ is the number of edges we have added to the vertex V in the dependent rounding phase d_v^* is the number of edges we have added during the threshold phase, where we have set the threshold as $\frac{2}{3}$ We can say that we have atlest a value greater than equal to $\frac{2}{3}$ for the x_v's for the vertices
Hence, we have that

$$3[\frac{\frac{2}{3}(d_v^* + a_v + f_v)}{k_v}] \leq 3x_v \leq 3OPT$$

Chapter 14

Elementary Theory of LP duality

14.1 Primal–Dual Framework

Primal–Dual framework has its origins in the design of exact algorithms. This framework has become a highly successful algorithm design technique over the last few years. Over the last few years many classical problems such as the *Facility Location Problem* have been approximated using the primal–dual schema and the original factor approximations have improved by leaps and bounds. In this lecture, I shall describe the primal–dual framework and describe some key properties of this schema. Then, I shall provide a few well known problems that we have already discussed and then we shall write out the dual of the primal programs.

14.1.1 A Quick Review

An LP in standard form is written as follows:

$$\min \sum_{j=1}^{n} e_j \cdot x_j$$

such that,

$$\sum_{j=1}^{n} a_{ij} \cdot x_j \geq b_i, \text{ for } i = 1, \ldots, m$$

and,

$$x_j \geq 0, \text{ for } j = 1, \ldots, n$$

The above standard form can be written more compactly as follows:

$$\min c^T \cdot x$$

such that,

$$A \cdot x \geq b$$

and,

$$x \geq 0$$

where we are given $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$ and we are solving the LP for $x \in \mathbb{R}^n$.

The dual of this LP is written in terms of dual variables y instead of primal variables x as follows:

$$\max \sum_{i=1}^{m} b_i \cdot y_i$$

such that,

$$\sum_{i=1}^{m} a_{ij} \cdot y_i \leq c_j, \text{ for } j = 1, \ldots, n$$

and,

$$y_i \geq 0, \text{ for } i = 1, \ldots, m$$

In compact form, the above dual LP can be written as:

$$\max\ b^T \cdot y$$

such that,

$$A^T \cdot y \leq c$$

and,

$$y \geq 0$$

Let us now write a familiar primal and the try to write its dual.

14.2 Cardinality Vertex Cover Problem as a Primal LP

Consider the CVC problem. The LP relaxation of the problem can be written as:

$$\min \sum_{i=1}^{n} x_i$$

such that,

$$x_i + x_j \geq 1, \text{ for each } \{i, j\} \in E$$

and,

$$x_i \geq 0, \text{ for } i = 1, \ldots, n$$

Now, let us try to write its **dual**:

$$\max \sum_{e \in E} y_e$$

such that,

$$\sum_{e:e \text{ incident on } i} y_e, \text{ for each } i \in V$$

and,

$$y_e \geq 0$$

14.2.1 An Interpretation of the above LP Dual

Consider the dual as a relaxation of an IP. To get the IP back, we replace $y_e \geq 0$ with $y_e \in \{0,1\}$. This IP is the *Maximum Matching Problem*.

The above LP dual has a combinatorial interpretation. However, note that not all LP duals have a combinatorial interpretation.

14.3 Properties of Duality

This section lays the foundation of the primal dual framework. In this section, we study two key properties of duality that we state as follows:

- Weak Duality Theorem
- Strong Duality Theorem

14.3.1 Weak Duality Theorem

Theorem 19 *If x is a feasible solution of the primal and y is a feasible solution of the dual then, $b^T \cdot y \leq c^T \cdot x$.*

In simpler terms, the above theorem is saying that:

"Dual is a lower bound on primal and primal is an upper bound on the dual."

The figure in Figure 14.1 shows a pictorial view of the above.

Proof:

$$\begin{aligned}
b^T \cdot y &= \sum_{i=1}^{m} b_i \cdot y_i \\
&\leq \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} \cdot x_j \right) \cdot y_i \\
&= \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} \cdot y_i \right) \cdot x_j \\
&\leq \sum_{j=1}^{n} c_j \cdot x_j \\
&= c^T \cdot x
\end{aligned}$$

□

Corollary 20 *The primal has a finite optimal solution iff the dual has a finite optimal solution.*

Next, we present *Complementary Slackness Conditions* with respect to the primal and dual LP's.

Complementary Slackness Conditions

The "complementary slackness conditions" state the following:

Let x and y be feasible solutions of the primal and dual LP's respectively. Then x and y are both optimal iff all the following conditions are satisfied:

- **Primal Complementary Slackness Condition**. Where, for each $j = 1, \ldots, n$, either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} \cdot y_i = c_i$.
- **Dual Complementary Slackness Condition**. Where, for each $i = 1, \ldots, m$ either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} \cdot x_j = b_i$.

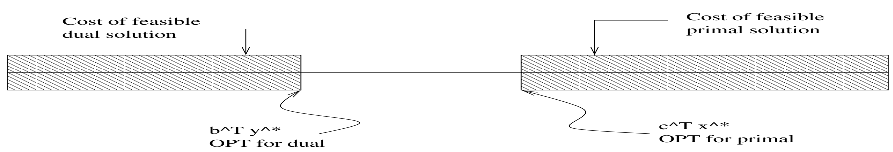


Figure 14.1: A Pictorial view of the Weak Duality Theorem.

14.3.2 Strong Duality Theorem

Theorem 21 *If x^*, y^* are primal and dual optimal solutions respectively then*

$$c^T \cdot x^* = b^T \cdot y^*$$

Revisiting CVC and Maximum Matching Problem in the context of the Strong Duality Theorem

Recall that the "weak duality theorem" stated that the OPT for Maximum Matching Problem (MMP) is a lower bound on the OPT for CVC. The "strong duality theorem" is stating that the OPT obtained from the relaxed version of both the problems are the same. To get a pictorial view, refer to Figure 14.2.

The case of bi–partite graphs yields the picutre in Figure 14.3.

The following theorem is due to König and Egervary.

Theorem 22 *The size of the a maximum matching in bi–partite graphs equals the size of a minimum* CVC.

14.4 The Maximum Flow–Minimum Cut Theorem

Max–Flow Problem:

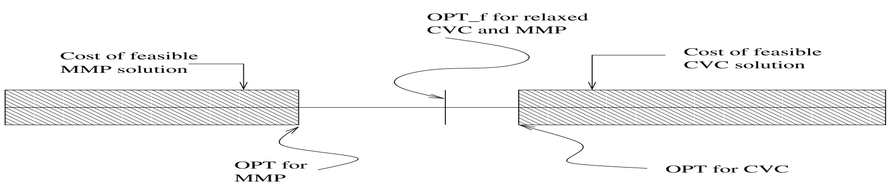


Figure 14.2: A Pictorial view of the Strong Duality Theorem as it applies to OPT and MMP.

Input: A digraph $G = (V, E)$ and edge capacities $c : E \to \mathbb{R}^+$ and a source s and a sink t.
Output: A flow $f : E \to \mathbb{R}^+$ such that:

1. Capacity Constraint: for each $(i, j) \in E, f(i, j) \leq c(i, j)$.
2. Flow Conservation: for each $i \in V \setminus \{s, t\}$

$$\sum_{j:(j,i)\in E} f(j,i) = \sum_{j:(i,j)\in E} f(i,j)$$

3. Maximality: The following is maximized:

$$\sum_{j:(s,j)\in E} f(s,j) - \sum_{j:(j,s)\in E} f(j,s)$$

14.4.1 An LP for Max–Flow Problem

$$\max \sum_{j:(s,j)\in E} f(s,j) - \sum_{j:(j,s)\in E} f(j,s)$$

such that,

$$f(i,j) \leq c(i,j), \forall (i,j) \in E$$

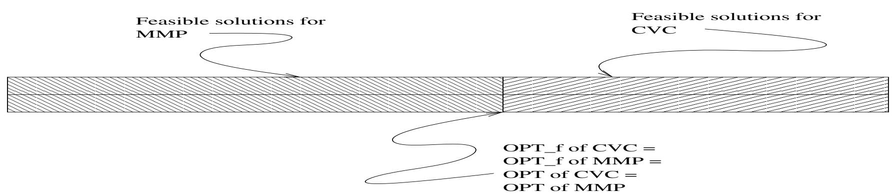


Figure 14.3: A pictorial view of the König–Egervary Theorem

and,

$$\sum_{j:(j,i)\in E} f(j,i) = \sum_{j:(i,j)\in E} f(i,j), \forall i \in V \setminus \{s,t\}$$

and,

$$f(i,j) \geq 0, \forall (i,j) \in E$$

A Simplification to the above LP

1. Add a directed edge (t,s) with $c(t,s) = \infty$ and enforce flow conservation at s and t. This will enable us to write out objective function as $\max f(t,s)$.

2. Replace the equality for conservation constraints by "$\leq$'s" everywhere where there are "$=$'s".

14.4.2 The Simplified LP

$$\max f(t,s)$$

such that,

$$f(i,j) \leq c(i,j), \forall (i,j) \in E$$

and,

$$\sum_{j:(j,i)\in E} f(j,i) - \sum_{j:(i,j)\in E} f(i,j) \leq 0, \forall i \in V$$

and,

$$f(i,j) \geq 0, \forall (i,j) \in E$$

14.4.3 The Dual of the above LP

- For each primal edge constraint, let p_{ij} be the corresponding dual variable.
- For each primal vertex constraint, let d_i be the corresponding dual variable.

The objective function is written as (as per the rules of the primal–dual framework):

$$\min \sum_{(i,j)\in E} p_{ij} \cdot c(i,j)$$

Now, look at the matrix representing the primal program:

$$\left(\begin{array}{cccccc}
1 & 0 & 0 & 0 & \dots & 0 \\
0 & 1 & 0 & 0 & \dots & 0 \\
0 & 0 & 1 & 0 & \dots & 0 \\
\vdots & \ddots & & & & \vdots \\
\vdots & & \ddots & & & \vdots \\
\vdots & & & \ddots & & \vdots \\
\vdots & & & & \ddots & \vdots \\
0 & 0 & 0 & 0 & \dots & 1 \\
\hline
\multicolumn{6}{c}{\dots} \\
\multicolumn{6}{c}{\ddots} \\
\multicolumn{6}{c}{+1\text{'s}, -1\text{'s and } 0\text{'s}} \\
\multicolumn{6}{c}{\ddots} \\
\multicolumn{6}{c}{\dots}
\end{array}\right)$$

In the above matrix (that corresponds to the primal LP), the submatrix above the horizontal line is an identity matrix that corresponds to the capacity constraints. The submatrix below the horizontal line contains elements in $\{0, \pm 1\}$. This submatrix corresponds to flow conservation constraints on all vertices. Since the matrix corresponds to the primal program, observe that a column corresponds to a primal variable representing an edge. The part of this column above the horizontal line will contain exactly one 1 and below the line, we will have exactly one $+1$ and exactly one -1 while the rest of the entries will be all zeros (0's).

We shall continue our discussion next time.

Chapter 15

The Primal-Dual Schema for approximation algorithms

We now have
maxf(t,s)
subject to

$$f(i,j) \leq c(i,j) \forall (i,j) \in E. - \mathbf{I}$$

$$\sum_{j:(j,i)\in E} f(j,i) - \sum_{j:(i,j)\in E} f(i,j) \leq 0 \forall i \in V. - \mathbf{II}$$

DUAL LP:
Let $d_i p$ be the dual variable corresponding to the Type I constraint for edge (i,j).
Let P_i be the dual variable for the Type II constraint for vertex i.
We want to minimize

$$\sum_{(i,j)\in E} c(i,j) \cdot d_{ij}$$

To obtain the dual constraints let us examine the primal constraint matrix.
Hence the dual constraint corresponding to primal variable f(i,j) are

$$d_{ij} - P_i + P_j \geq 0 \forall (i,j) \in E$$

And dual constraint corresponding to f(t,s) are

$$P_s - P_t \geq 1$$

$$d_{ij} \geq 0 \forall (i,j) \in E$$

$$P_i \geq 0 \forall i \in V$$

Hence Dual LP

$$min \sum (i,j) \in E \cdot c(i,j).d_{ij}$$

subject to

$$d_{ij} - P_i + P_j \geq 0 \forall (i,j) \in E$$

$$P_s - P_t \geq 1$$

$$d_{ij} \geq 0 \forall (i,j) \in E$$

$$P_i \geq 0 \forall i \in V$$

Consider the IP obtained by replacing $d_{ij} \geq 0$ by $d_{ij} \in$ {0,1} and $P_i \geq 0$ by $P_i \in$ {0,1}
Observe that the above LP is a relaxation of this IP.
$*$ How is this IP interpretted ?
In any feasible solution $P_s = 0$ and $P_t = 0$.
Let V_0 = { i $\in$ V | $P_i = 0$ }
V_1 = { i $\in$ V | $P_i = 1$ }
In an optimal solution $d_{ij} = 0$ $\forall$ (i,j) $\in$ E and { (i $\in$ V_0 and j $\in$ V_0) or (i $\in$ V_1 and j $\in$ V_1)}

$$d_{ij} = 0 \forall (i,j) \in E : i \in V_0 and j \in V_1$$

$$d_{ij} = 1 \forall (i,j) \in E : i \in V_1 and j \in V_0$$

Hence the objective function is minimizing the total capacity of edges from V_1 to V_0

Primal Dual Schema For Approximation Algorithms

Consider the following approximate complementary slackness coditions:

Approximate Primal Complementary Slackness

For each j = 1,2,....., n $x_j = 0$
OR

$$\frac{C_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} \cdot y_i \leq C_j where \alpha \geq 0$$

Approximate Dual Complementary Slackness

For each i = 1,2,......, m $y_i = 0$
OR

$$\beta \cdot b_i \geq \sum_{j=1}^{n} a_{ij} \cdot x_j \geq b_i where \beta \geq 0$$

Claim: Let x and y be feasible primal and dual solutions satisfying all of the above constraints. Then

$$\sum_{j=1}^{n} c_j \cdot x_j \leq \beta \cdot \alpha \sum_{i=1}^{m} b_i \cdot y_i$$

Proof:

$$\sum_{j=1}^{n} c_j \cdot x_j \leq \sum_{j=1}^{n} (\alpha \cdot \sum_{i=1}^{m} a_{ij} \cdot y_i) \cdot x_j$$

$$= \alpha \cdot \sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} \cdot x_j) \cdot y_i$$

$$\leq \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i \cdot y_i$$

Using this in Set Cover
Set Cover LP Relaxation

$$min \sum_{j=1}^{n} c(s_j) \cdot s_j$$

Subject to

$$\sum_{j:u \in S_j} x_j \geq 1 for each element i = 1, 2, ..., m$$

$$x_j \geq 0 for each j = 1, 2, ..., n$$

Dual LP

$$max \sum_{i=1}^{m} y_i$$

Subject to

$$\sum_{i \in S_j} y_i \leq c(S_j) for each j = 1, 2, ..., n$$

$$y_i \geq 0 for all i = 1, 2, ..., m$$

Let us state the Approximate Complementary Slackness Conditions with $\alpha = 1$ and $\beta =$ f, where f = max. frequency of any element i.
Approximate Primal Complementary Slackness

For each set S_j , $x_j = 0$ or

$$\sum_{i \in S_j} y_i = C(S_j)$$

Approximate Dual Complementary Slackness
For each element i = 1,2,.....,m , $y_i = 0$ or

$$\sum_{j: i \in S_j}^{n} x_j \geq 1$$

If we can come up with an integral feasible solution x and a dual solution y satisfying the slackness conditions mentioned, then we get a factor-f approximation algorithm for Set Cover.

How do we find such x and y ? First let us restate the primal complementary slackness conditions: It is saying that for each set S_j , j=1,2,....,n, we cannot have x_j ¿ 0 and

$$\sum_{i \in S_j} y_i < c(S_j)$$

≡ For each set S_j , j=1,2,.....,n, if x_j ¿ 0 then

$$\sum_{i \in S_j} y_i = c(S_j)$$

1. Start with x = 0 (integral non-feasible Primal solution) and y = 0 (feasible dual solution)
2. At each step we make x more feasible maintaining integrality.
3. At each step make y more optimal.
4. At all steps approximate complementary slackness conditions are maintained.

Chapter 16

Approximation of Set Cover via the primal-dual schema

A factor-f algorithm for SET COVER via the primal-dual framework
The primal of this problem, which is the LP-relaxation for SET COVER is the following:

Minimize

$$\sum_{j=1}^{n} x_j \cdot c(S_j)$$

subject to

$$\begin{aligned}\sum_{j:i\in S_j} x_j &\geq 1 \text{ for each } i = 1, 2, \cdots, m\\ x_j &\geq 0 \text{ for each } j = 1, 2, \cdots, n\end{aligned}$$

The dual of this problem is:
Maximize

$$\sum_{i=1}^{m} y_i$$

subject to

$$\begin{aligned}\sum_{i\in S_j} y_i &\leq c(S_j) \text{ for each } j = 1, 2, \cdots, n\\ y_i &\geq 0 \text{ for each } i = 1, 2, \cdots, m\end{aligned}$$

The *primal complementary slackness* condition is:
For each $j = 1, 2, \cdots, n : x_j = 0$ or $\sum_{i\in S_j} y_i = c(S_j)$
The *dual complementary slackness* condition is:
For each $i = 1, 2, \cdots, m : y_i = 0$ or $\sum_{j:i\in S_j} x_j = 1$
The corresponding *approximate primal complementary slackness* condition is:

$$\frac{c(S_j)}{\alpha} \leq \sum_{i\in S_j} y_i \leq c(S_j)$$

The corresponding *approximate dual complementary slackness* condition is:

$$\beta \geq \sum_{j:i\in S_j} x_j \geq 1$$

(Note that $\alpha = 1$ and $\beta = f$ gets us the original "exact" constraints.)
We would like these two approximate constraints to be maintained. If we can produce x and y such that x is a feasible, integral, primal solution and y is a feasible dual solution satisfying these approximate constraints, then x is a factor-f approximation solution for SET COVER.

Remarks on the approximate constraints
Approximate Dual Constraint:
How hard is it to maintain the dual constraint? Easy. (It comes for free and is always satisfied.) If x is a feasible integral solution, then the approximate dual complementary slackness condition is satisfied.
Approximate Primal Constraint:
Another way to write this condition is as follows:
For each $j = 1, 2, \cdots, n : x_j \neq 0 \Rightarrow \sum_{i\in S_j} y_i = c(S_j)$
This suggests a way of setting x_j's to 1's: when a set S_j becomes "tight" (ie. $\sum_{i\in S_j} y_i = c(S_j)$) then set the corresponding $x_j = 1$

Algorithm

1. Set $x = 0$ (integral, infeasible primal solution) and $y = 0$ (feasible dual solution).
   Note that approximate primal complementary slackness condition is satisfied. The approximate dual complementary slackness condition is NOT satisfied after the initial step, after we start increasing y_i's. We do not worry about this because as soon as x becomes feasible, the dual constraint will be re-satisfied.

2. Pick an uncovered element i. Increase y_i to the minimum value such that some set containing i is tight.

3. For all sets S_j that are tight, set $x_j = 1$ (ie. throw set S_j into solution).

4. Remove all elements i covered by sets in solution. (This simply means their current y_i's cannot be increased any further.) Go back to step 2.

In step 2, suppose we increased y_i's "synchronously". The first tight set is a set S_j with minimum $\frac{cost(S_j)}{|S_j|}$. This is equivalent to our greedy choice in the greedy algorithm approach.

Steiner Tree
Input: A graph $G = (V, E)$ with edge costs $C : E \to Q^+$ and a set $R \subseteq V$ of *required* vertices.
Output: A tree in G with minimum cost containing R. (If R is a tree, this becomes the minimum spanning tree problem which we know how to solve in P. The generalization is in NP. The hard part comes from choosing which vertices not in R should participate in the solution.)
Status: Easy factor-2 approximation algorithm via minimum spanning tree. The approximation

factor has been improved many times in the last decade (eg. 5/3-factor, all subsequent lower factors are due to Zelikovsky).

There is a specific version of this problem called *Euclidean Steiner Tree*
Input: Points in $\Re^n$
Output: A tree with smallest cost connecting these points, but may include other points as well.
Status: There is a PTAS for this (due to S. Arora).

Solving the Steiner Tree problem

1. Reduce the problem to the Metric Steiner tree problem.
   Specifically, construct the following:

   $$G = (V, E) \rightarrow G^M = (V, E^M)$$

   where G^M is the complete graph and $c(u, v)$ = cost of cheapest path between u and v in G.

2. Solve the Steiner Tree problem on G^M and R. This is called the *Metric Steiner Tree* problem. (The edge costs of G^M satisfy triangle inequality.)

   Lemma 23 *Cost of optimal Steiner tree of R in G = cost of optimal Steiner tree of R in G^M.*

   Proof: This is clear from the fact that for any edge (u, v), its cost in G^M is no more than its cost in G. So we might as well solve the problem on G^M. □

3. Compute a minimum spanning tree T of G^M.

   Lemma 24 *Cost of T, $cost(T) \leq 2 \cdot OPT$*

   Proof: Consider an inorder traversal or tour of the edges in the optimal Steiner tree. When backtracking, we skip the vertices that we have already traversed by adding an edge to an unvisited vertex in our graph. So it is clear that we can, at most, end up doubling our edges. These shortcut edges do not increase the cost of the tour because of triangle inequality. Hence, the cost of our tour $\leq 2 \cdot OPT$. Remove one edge in this cycle and we get a path $\leq 2 \cdot OPT$. So if we used the minimum spanning tree, it would have to be less than this. □

Steiner Forest
The algorithm we describe is by Goemans and Williamson (factor-2 approximation, best known).
Input: A graph $G = (V, E)$ with edge costs $C : E \rightarrow Q^+$. A collection of subsets of V, $S_1, S_2, \cdots, S_k$
Output: A subgraph of G with minimum cost such that for any S_i, vertices in S_i lie in the same connected component of the subgraph.

Chapter 17

Steiner Forest via the primal-dual schema

Steiner Forest

We will obtain a 2-approximation algorithm for the steiner forest problem using the primal-dual schema, with the idea of growing duals in a synchronized manner.

Definition Given an undirected graph $G = (V, E)$, a cost function c on the edges, and a collection of disjoint subsets of V, find a minimum cost subgraph in which each pair of vertices belonging to the same subset are connected.

Input

An undirected graph $G = (V, E)$,
a cost function on the edges, $c : E \to Q^+$, and
a collection of subsets of V, $\{S_1, S_2, \cdots, S_k\}$

Output

A minimum cost subgraph F of G such that $\forall u, v \in S_i, \exists$ a $u - v$ path in F.

Notation

1. Define the connectivity requirement of G be a function r, the set of all unordered pair of vertices to $\{0, 1\}$, such that

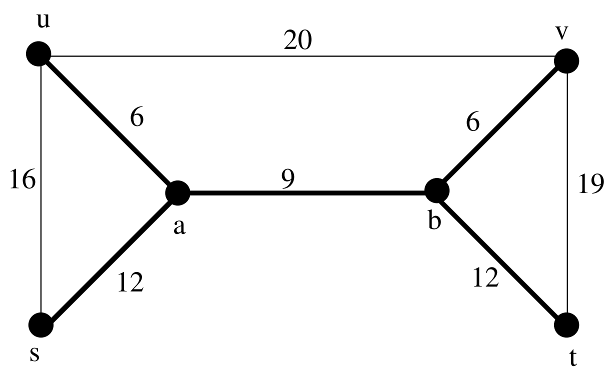
$$r(u,v) = \begin{cases} 1 & \text{if } u, v \in S_i \text{ for some } i \\ 0 & \text{otherwise} \end{cases}$$

2. Recall that a cut in G is a partition $(S, \overline{S})$ of V. We will use S to denote such a cut. Define a fuction f such that,

$$f : 2^V \to \{0, 1\}$$

as follows :

$$f(S) = \begin{cases} 1 & \text{if } \exists u \in S, v \in \overline{S} \text{ such that } r(u,v) = 1 \\ 0 & \text{otherwise} \end{cases}$$



Example 1. Let G be the above graph, with vertices {u, v, a, b, s, t}, and edge weights as displayed on the edges.
Let S1 = {u, v} and S2 = {s, t} be the two subsets of V. Then a feasible soultion is the subgraph defined by the bold edges.

Figure 17.1: A graph and a feasible subgraph

Observation For any feasible solution F, $f(S) \leq$ number of edges of F that cross the cut $(S, \overline{S})$. Notice that the converse also holds. i.e., For any subgraph F of G, if for every cut $S \subseteq V$, $f(S) \leq$ number of edges of F crossing $(S, \overline{S})$, then F is a feasible solution to the problem.

Since the converse also holds, we have an alternate charecterization. We can write the optimization problem as :

A min-cost subgraph F of G such that $\forall S \subseteq V$, $f(S) \leq$ number of edges of F crossing $(S, \overline{S})$.

IP formulation and LP relaxation Let x_e be an indicator variable for each edge $e \in E$. The objective function is :

$$min \sum_{e \in E} x_e c(e)$$

subject to,

$$\sum_{e: e \in \delta(S)} \geq f(S) \qquad \forall S \subseteq V$$

$$x_e \in \{0, 1\}; \qquad \forall e \in E$$

where, $\delta(S)$ denotes the set of edges crossing the cut $(S, \overline{S})$.

The LP relaxation is obtained by relaxing $x_e \in \{0, 1\} by x_e \geq 0$.

Dual of the LP relaxation Let Y_S denote the dual variable corresponding to cut S. The dual of the primal LP defined above is :

$$max \sum_{S \subseteq V} Y_S f(S)$$

subject to,

$$\sum_{S:e\in\delta(S)} Y_S \leq c(e) \qquad \forall e \in E$$

$$Y_S \geq 0 \qquad \forall S \subseteq V$$

Complementary Slackness Conditions The primal and dual complementary slackness conditions are defined as follows :

Primal:

$\forall$ edge $e \in E$, $x_e \neq 0 \Rightarrow \sum_{S:e\in\delta(S)} Y_S = c(e)$.

We wil use the exact version of the primal complementary slackness condition. i.e., we set $\alpha = 1$

Dual :

$\forall$ cuts $S \subseteq V$, $Y_S \neq 0 \Rightarrow \sum_{e:e\in\delta(S)} x_e = f(S)$.

The approximate version of the dual complementary slackness condition is :

$\forall$ cuts $S \subseteq V$ $Y_S \neq 0 \Rightarrow f(S) \leq \sum_{e:e\in\delta(S)} x_e \leq 2f(S)$.

i.e., $\beta = 2$.

If we can get an integral primal feasible solution, and a dual feasible solution satisfying these constraints, that would imply a factor-2 algorithm for the steiner forest problem. However, we don't know how to satisfy the dual complementary slackness condition.

Algorithm rough sketch

1. Start with $x_e = 0$, $\forall e \in E$ and $Y_S = 0$, $\forall S \subseteq V$
   ($x_e = 0$ represents an infeasible, integral primal solution; and $Y_S = 0$ represents a feasible dual solution).

2. Increase the Y_S's until some edge e becomes tight.
   i.e., $\sum_{S:e\in\delta(S)} Y_S = c(e)$. Throw e into the solution ($x_e = 1$).

Features of the algorithm

1. Y_S values are increased in a synchronous fashion.

2. In each iteration, we will increase Y_S for a small number of cuts S.

   - Define a cut S to be *unsatisfied* if $f(S) = 1$, but no edge in the currently chosen set crosses S.
   - Define an *active cut* as a minimal unsatisfied cut (with respect to inclusion).
     In each iteration, we increase Y_S synchronously $\forall$ active cut S.

Claim : A cut S is active iff it is a connected component in the current subgraph and $f(S) = 1$. This claim will be proved later on.

Algorithm

1. Set $F' = \phi$ (the set of chosen edges; corresponds to saying $x_e = 0\ \forall e \in E$)

2. while($\exists$ an unsatisfied cut) do

   Increase Y_S value synchronously for every active cut S, until some edge e becomes tight.

   $F' \leftarrow F' \bigcup \{e\}$.

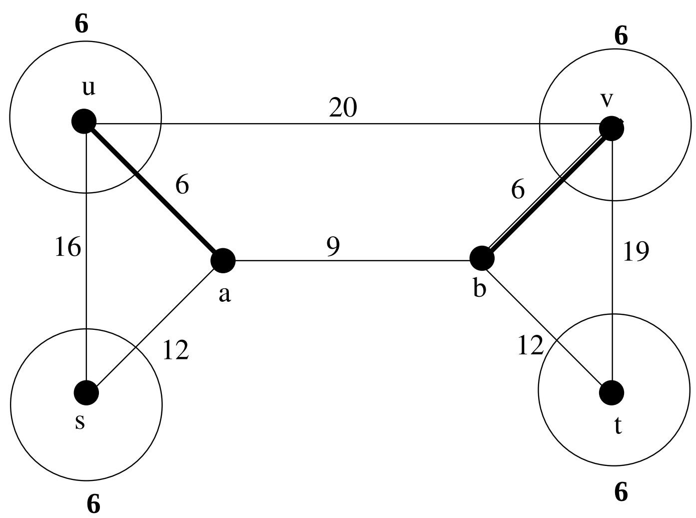
3. endwhile

4. Prune the redundant edges from F'. i.e., delete every edge e from F' such that $F' - \{e\}$ is feasible.

Claim : F' is a primal feasible solution, and y is a dual feasible solution.

Proof: Before the pruning step, F' satisfies all connectivity requirements (because there are no unsatisfied cuts). F' is a forest because of the property of active cuts. edges that become tight, and are added in any iteration connect one connected component to another.

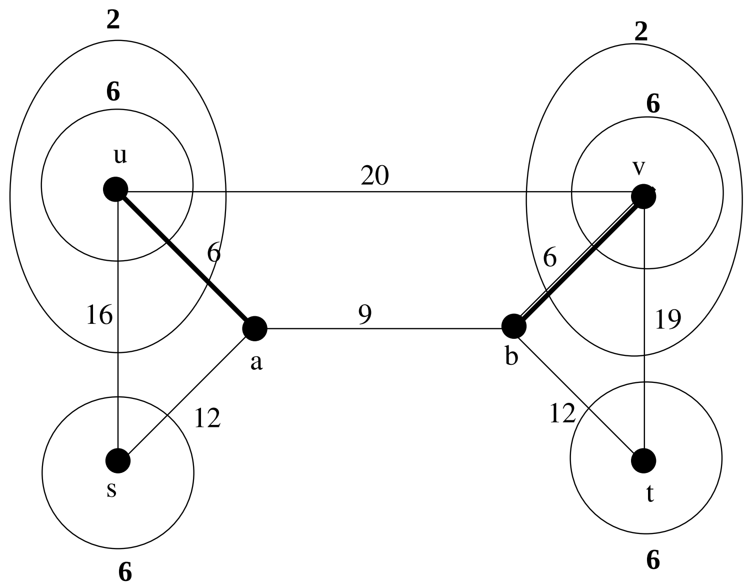
Therefore, for any u, v such that the connectivity requirement is 1, ther is a unique uv path. Hence, all edges in the path are non-redundant and hence not deleted. Hence, after the pruning step also, connectivity is maintained. $\square$



In the begining of the iteration, the active sets are {s}, {t}, {u}, {v}. When their dual variables are raised by 6 each, the edges {u,a} and {b,v} go tight. One of them, say {u,a} is picked, and in the next iteration, {u,a} replaces the active set {u}.In the next iteration, without having to raise any of the varaibles, the edge {b,v} becomes tight, and {b,v} replaces the active set {v}. The edges that are picked are marked in bold.

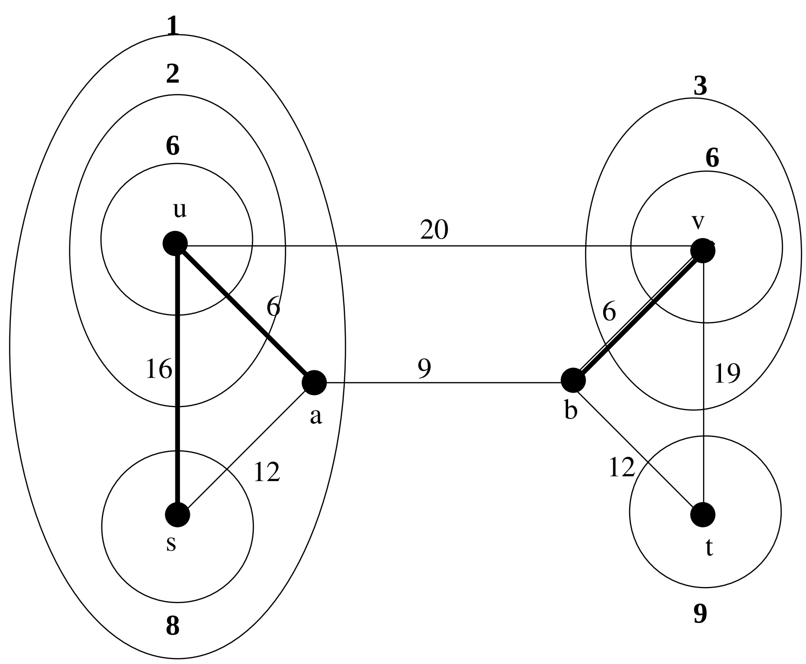
Figure 17.2: First instance when the Y_S values become tight

Example execution of the algorithm Consider the graph and the subsets of the vertices as in example 1. The following figures show the execution of the algorithm on the above graph.



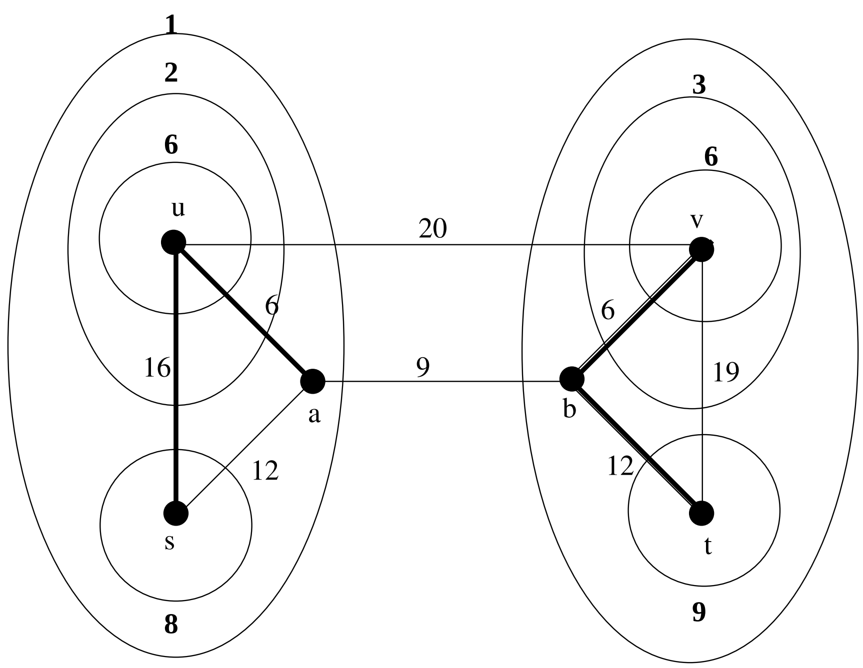
When the Y_S values are increased by 2, the edge {u,a} becomes tight, and the active sets are now : {u,s,a}, {v,b}, {t}.

Figure 17.3: Instance when the Y_S values become tight



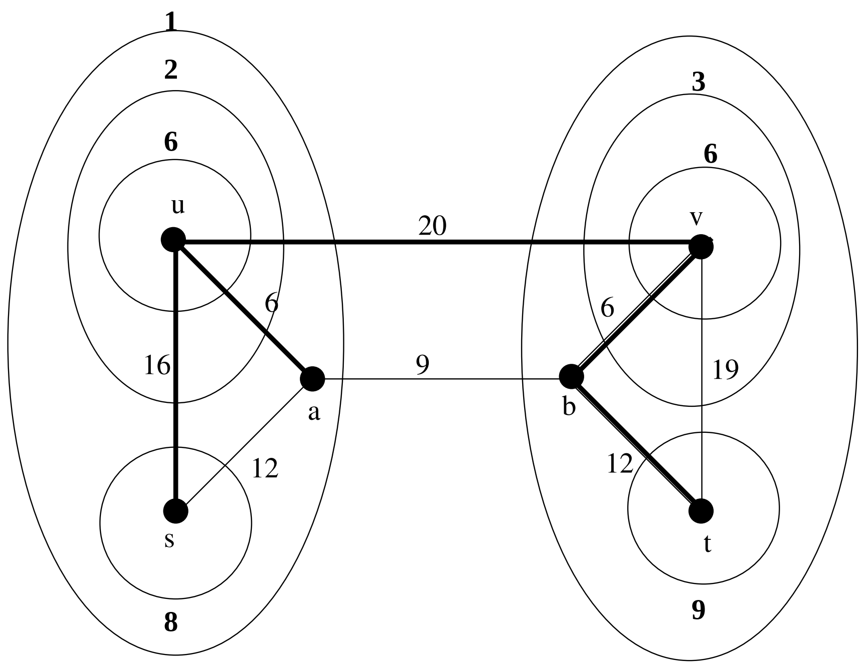
At the next iteration, the edge {b,t} becomes tight. The active sets are {u,s,a}, {v,b,t}

Figure 17.4: Instance when the Y_S values become tight



At the next iteration, the edge {u,v} becomes tight

Figure 17.5: Instance when the Y_S values become tight



The subgraph produced by the algorithm consits of the edges {u,a}, {v,b}, {u,a}, {b,t}, {u,v}

Figure 17.6: Final output of the algorithm

Chapter 18

Steiner Forest Algorithm (continued)

Analysis of the primal-dual Steiner forest algorithm (cont.).

Claim 2 A cut S is active iff S is a connected component w.r.t the current set of chosen edges and $f(S) = 1$.

Proof:

($\Leftarrow$) If S is a connected component and $f(S) = 1$, then S is unsatisfied. Furthermore, S is minimal, because any proper subset $S' \subset S$ has an edge going out of S'.

($\Rightarrow$) Suppose S is active but S is not a connected componet, clearly, no currently chosen edge crosses $(S, \bar{S})$. Hence S is the union of two or more connected componets. Since S is active,$f(S) = 1$. Hence for some $u \in S$ and $v \in \bar{S}$, $r(u, v) = 1$. Suppose $u \in C$ for some connected component C in S, then C is unsatisfied, implying S is not a minimal unsatisfied cut, a controdiction. $\square$

Claim 3 $\sum_{e \in F'} C_e \leq 2 \cdot \sum_{S \in V} Y_S \cdot f(S)$

Proof:

$$\sum_{e \in F'} C_e = \sum_{e \in F'} (\sum_{S : e \in \delta(S)} Y_S) = \sum_{S \subseteq V} (\sum_{e : e \in \delta(s) \cap F'} Y_S) = \sum_{S \subseteq V} Y_S \cdot |\delta(S) \cap F'| = \sum_{S \subseteq V} Y_S \cdot deg_{F'}(S)$$

where $deg_{F'}(S) = |\delta(S) \cap F'|$, denoting the number of edges in F' that cross $(S, \bar{S})$, which has no relation to Y_S.

We need to show that

$$\sum_{S \subseteq V} Y_S \cdot deg_{F'}(S) \leq 2 \cdot \sum_{S \in V} Y_S \cdot f(S)$$

We will show something stronger, that is,

$$Changes\ in\ L.H.S \leq Changes\ in\ R.H.S.$$

Initially, L.$H.S = R.H.S. = 0$. Consider any arbitrary iteration and let Δ be the increate in Y_S, during that iteration,

$$Changes\ in\ L.H.S = \sum_{active\ S} deg_{F'}(S) = \Delta \cdot \sum_{active\ S} (S)$$

$$Changes\ in\ R.H.S. = 2 \cdot \sum_{active\ S} \Delta \cdot f(S) = 2 \cdot \Delta \cdot (\ number\ of\ active\ cuts\ S)$$

We want to show that

$$\sum_{activeS} deg_{F'}(S) \leq 2 \cdot \Delta \cdot (number\ of\ active\ cuts\ S)$$

that is, the average degree of active cuts w.r.t. F':

$$\frac{\sum deg_{F'}(S)}{number of active S} \leq 2$$

To finish the proof, we need one additional claim.

Claim 4 Let C be a component w.r.t. the currently chosen set of edges such that $f(C) = 0$, then $deg_{F'}(C) \neq 1$.
Proof: Suppose the claim is false, that is $f(C) = 0$ but $deg_{F'}(C) = 1$. So there exists a unique $e \in F'$ that crosses $(C, \bar{C})$.
Since $e \in F' \Rightarrow e$ is not redundant w.r.t. F'.
$\Rightarrow e$ is an edge on a unique $u - v$ path for some u,v, and $r(u, v) = 1$
$\Rightarrow$ W.l.o.g. $u \in C and v \in \bar{C}$.
$\Rightarrow f(C) = 1$ a controdiction. □

Claim 4 tells us that any inactive component C has $deg_{F'}(C) = 0$(i.e. it is isolated) or $deg_{F'}(C) \geq 2$. From this observation, the result follows. □

To show that the analysis is tight for this algorithm. Consider the following example:
$V = \{1, 2, 3, ..., n, (n + 1)\}(1, 2, ..., (n + 1)$ are labels on vertices), where $1, 2, 3, ..., n \in K_n$ and edges in K_n cost 2 each, edges from $(n+1)$ to each vertex in K_n have unit cost. And $S_1 = \{1, 2, ...n\}$.
The $OPT =$ n. Cost of solution is $2 \cdot (n - 1)$.

Upper Bound on Integrality Gap Let OPT_f denote the optimal solution for primal problem.

$$OPT \leq \sum_{e \in F'} C_e \leq 2 \cdot \sum y_S \cdot f(S),\ and$$

$$\sum y_S \cdot f(S) \leq OPT_{dual} = OPT_f$$

$$\Rightarrow \frac{OPT}{OPT_f} \leq 2$$

thus giving a upper bound on integrity gap.

Lower Bound on Integrality Gap Consider a cycle on n vertices, with all edges of cost 1. The cost of dual solution found by algorithm is $\frac{n}{2}$, which is the optimal for the dual because there is a primal feasible solution with cost $\frac{n}{2}$. Therefore,

$$OPT_f = \frac{n}{2}, OPT = (n-1)$$

$$\Rightarrow \frac{OPT}{OPT_f} \text{ (is essentially) } \geq 2$$

We will discuss "Facility Location Problem" nextly.

Facility Location Problem

Input:A set C (of cities), a set F (of facilities). The cost of opening facility $i \in F$ is f_i. The cost of servicing a city $j \in C$ using a facility $i \in F$ is C_{ij}.

Output: A set $I \subseteq F$ of open facilities and a function $\Phi : C \to I$ such that total cost

$$(\sum_{i \in I} f_i + \sum_{j \in C} C_{\Phi j})$$

is minimized.

We will discuss a factor-3 approximation algorithm using the primal-dual schema.

Chapter 19

Facility Location via the primal-dual schema

IP for metric facility locatin problem

Indicator variable $y_i \in \{0,1\}$ indicates if facility $i \in F$ is open
Indicator variable $x_{ij} \in \{0,1\}$ indicates if city $j \in C$ is connected to facility $i \in F$.

$$\text{minimize} \sum_{i \in F} y_i \cdot f_i + \sum_{i \in F, j \in C} x_{ij} \cdot c_{ij}$$

s.t.

$$\sum_{i \in F} x_{ij} \geq 1 \text{ for each } j \in C$$

$$y_i - x_{ij} \geq 0 \text{ for each } i \in F, j \in C$$

$$y_i \in \{0,1\}, x_{ij} \in \{0,1\} \text{ for each } i \in F, j \in C$$

And the LP-relaxation for this problem is obtained by replacing $y_i \in \{0,1\}, x_{ij} \in \{0,1\}$ by $y_i \geq 0, x_{ij} \geq 0,\ \forall i \in F, j \in C$

The dual problem of LP-relaxation

Indocator variables α_j, β_{ij}

$$\text{maximize} \sum_{j \in C} \alpha_j$$

s.t.

$$\sum_{j \in C} \beta_{ij} \leq f_i \text{ for each } i \in F$$

$$\alpha_j - \beta_{ij} \leq c_{ij} \text{ for each } i \in F, j \in C$$

$$\alpha_j \geq 0, \beta_{ij} \geq 0 \text{ for each } i \in F, j \in C$$

Complemetary slackness conditions

For primal problem
(1) For each $i \in F, y_i > 0 \Rightarrow \sum_{j \in C} \beta_{ij} = f_i$
(2) For each $i \in F, j \in C, x_{ij} > 0 \Rightarrow \alpha_j - \beta_{ij} = c_{ij}$

For dual problem
(3) For each $j \in C, \alpha_j > 0 \Rightarrow \sum_{i \in F} x_{ij} = 1$
(4) For each $i \in F, j \in C, \beta_{ij} > 0 \Rightarrow y_i = x_{ij}$

Interpretation of these conditions is:

Assume that we have an integral primal feasible solution (X, Y). This induces a set I of open facilities and an assignment Φ of each city to a facility. So let (α, β) be a feasible dual solution, soppose (X, Y) and (α, β) satisfy the complemetary slackness conditions, then

(1) a facility $i \in F$ is open only when the cities contribute enough towards opening the facility.

(2) a connection from j to i is established only if city j pays enough so that after its contribution towards opening facility i has been subtracted, there is enough to pay for the connection cost to i.

(3) for a integral solution, it is trivial

(4) if a city $j \in C$ makes a positive contribution towards opening a facility $i \in F$, and j is open then j is connected to i.

Approximation of primal complementary slackness conditions

$$y_i > 0 \Rightarrow \frac{f_i}{3} \leq \sum_{j \in C} \beta_{ij} \leq f_i$$

$$x_{ij} > 0 \Rightarrow \frac{c_{ij}}{3} \leq \alpha_j - \beta_{ij} \leq c_{ij}$$

The following algorithm actually maintains conditions (1), (3) and (4), only (2) is relaxed as showed above.

Algorithm
Phase 1

1. $\alpha_j = 0, \beta_{ij} = 0, \forall i \in F, j \in C, I = \emptyset$
2. increase α_j synchronously for all cities $j \in C$

- at a point where for some edge $(i, j), \alpha_j = c_{ij}$, such a edge is said to be tight.
- once an edge (i, j) becomes tight, any further increase in α_j implies an increase in β_{ij} at the same rate. (so $\alpha_j - \beta_{ij} = c_{ij}$ is maintained)
- at some point for some facility $i, \sum_{j \in C} \beta_{ij} = f_i$, i is said to be temporarily open. (Now β_{ij}'s can no longer increase $\Rightarrow \alpha_j$'s for j's that have tight edges to i also can not increase.)

- once a facility $i \in F$ is open, any unconnected city $j \in C$ s.t. (i, j) is tight is connected to i, i is said to be the connecting witness for j.

This terminates when all cities are connected.

Observation: a city $j \in C$ can make positive contributions to several facilities $\Rightarrow$ condition (4) could be violated.

Phase 2

Let F_t be the set of temporarily open facilities, let H be a graph with $V(H) = F_t$ and $E(H) = \{(i, i')|i \in F_t, i' \in F_t, and \exists\ j \in C : \beta_{ij} > 0$ and $\beta_{i'j} > 0\}$

Let I be a maximal independent set in H, and this is our set of permanently open facilities. Now we define Φ:

For each $j \in C$, define $X_j = \{i \in F_t|\beta_{ij} > 0\}$, and $|I \cap X_j| \leq 1$. Now:

(1) if $|I \cap X_j| = 1$, that is, there is an open facility i to which j makes a contribution, set $\Phi(j) = i$

(2) if city $j \in C$ has not been connected in step (1), consider i', the connecting witness of j, if $i' \in I$, set $\Phi(j) = i'$

(3) if $i' \notin I$, i' has some neighbor in H that is open (since I is a maximal independent set), let $''$ be an arbitrary open neighbor of i', set $\Phi(j) = i''$

Chapter 20

Facility Location (continued)

From F_t get I, which are permanently open facilities by finding maximal independent set. Then we have to determine the connection ϕ.

Consider a city $j \in C$, $Y_j = \{i \in F | B_{ij} > 0\}$.

1. If $|X_j \cap I| = 1$ then set $\phi(j) = i$ where $i \in X_j \cap I$.

2. Otherwise, let i' be the connecting witness for j. $i' \in I$ then set $\phi(j) = i'$.

3. Otherwise (i.e. $i' \in I$) there is a neighbor (in H) of i' in I. Call this neighbor i'' and set $\phi(j) = i''$.

Claim: The dual feasible solution (α, β) and the integral primal feasible solution (I, ϕ) satisfy the slackness condition listed.

Proof: Suppose that y_i and x_{ij} denote the feasible solution after phase 2.

(1) y_i implies $i \in I$. Since $I \in F_t$ and F_t only contain facility $i \in F$ s.t. $\sum \beta_{ij} = F_t$ and furthermore, since β_{ij} do not increase once $i \in F_t$. Therefore this is true.

(3) is trivial

(4) Suppose that for some $i \in F, j \in C : \beta_{ij} > 0$. If $y_i = 0$, then clearly $x_{ij} = 0$.

If $y_i = 1$, then is must be the case that for any $i' \in F$ s.t. $\beta_{ij} > 0$, then $i' \notin I$.

$\Rightarrow \phi(j)$ is set to i in step 1. Therefore $x_{ij} = 1$.

(2) Consider $i \in F, j \in C : x_{ij} > 0$. This implies that $x_{ij} = 1$.

(i.e.) city i is the connected to facility j.

This connection can be in (1˜3) of the steps defining ϕ. If the connections made in (1) then $\alpha_{j\_}\beta_{ij} = C_{ij}$. When the connection is made in (3), there are some other cities that are making positive contribution to i'. We need to show that : $\alpha_j \geq C_{ij}/3$.

(Figure 1 comes into this part.)

ftbpF3.8579in2.4829in0ptfig1.gif

Assume that we increase α_j simultaneously, say that i was thrown into F_t at time t_1 and i' and t_2.

Claim: $\alpha_i \geq t_2 \Rightarrow \alpha_j \geq \alpha_{j'}$.

$\alpha_j \geq C_{i'j}$ because $\alpha_j - \beta = C_{ij'}$. $\alpha_{j'} > C_{ij'}$. These imply $\alpha_j > C_{i'j'}$ and $\alpha_j > C_{ij'}$.

Also $\alpha_j \geq C_{i'j} \Rightarrow 3\alpha_j \geq C_{ij} + C_{i'j} + C_{i'j'} \geq C_{ij}$.

Therefore $\alpha_j \geq C_{ij}/3$.

Next topic is approximation algorithm using Semidefinite-Programming by goemans and Williamson 1995.

Example: Max-Cut

Imput: A graph $G = (V, E)$

Outout: A partition of V into $(S, \overline{S})$ s.t. sum of crossing edge weights is maximized.

Suppose for each vertex $i \in V$, we have a variable $y_i \in \{\pm 1\}$, where $y_i = 1$ if $i \in S$. $y_i = -1$ otherwise. And $S \cup \overline{S} = V$.

$\rightarrow i, j$ in the same set then $y_i y_j = 1$ and $(1 - y_i y_j)/2 = 0$.

$\rightarrow i, j$ not in the same set then $y_i y_j = -1$ and $(1 - y_i y_j)/2 = 1$.

We would like to solve the following program.

max $\sum_{(i,j)} \frac{(1-y_i y_j)}{2} w_{ij}$ where $y_i \in \{i, j\}$ or $y_i^2 = 1$. This is a strict quadratic program. To relax this problem, we solve

max $\sum_{(i,j)} \frac{(1-v_i v_j)}{2} w_{ij}$ where v_i are n-dimensional vectors s.t. $v_i v_i = 1$.

Chapter 21

Max-Cut via Semidefinite Programming

MAX CUT
Input:A graph $G = (V, E)$ with edge weights $W : E- > Q^+$
Output:A partition $(S, \bar{S})$ such that the sum total of the weights of edges crossing $(S, \bar{S})$ is maximized

Applications: MAX-CUT has a lot of applications in VLSI design,also in graph partitioning for solving partial differential equations(variant of MAX-CUT)

Let us analyze what happens if we throw each vertex into S or $\bar{S}$ independently with probability $\frac{1}{2}$ into the solution.We get a factor $\frac{1}{2}$ approximation algorithm(in the expected sense)

$$Prob[edge\{u, v\}crosses(S, \bar{S})] = \frac{1}{2}$$

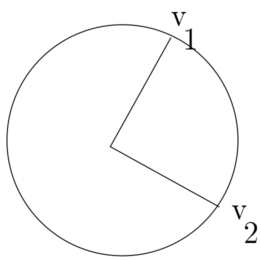
Therefore the expected contribution of edge $\{u, v\}$ is $\frac{W_v}{2}$. Expected cost of the solution $= \frac{1}{2}\sum_{\{u,v\}\in E} W_{uv} \geq \frac{1}{2}OPT$. This can be easily derandomized using the method of conditional expectation

Algorithm using SDF:

We use a technique called semi-definite programming to solve this problem.This technique was invented by Goemans and Williamson in 1995
The following program solves MAX-CUT

$$max \sum_{\{i,j\}\in E} W_{ij}\frac{(1 - y_i y_j)}{2}$$

such that ${y_i}^2 = 1$
We can observe that this is a quadratic program because sum of the terms in the objective function



and constriants are quadratic.Solving quadratic programs is NP-Complete

Consider the following vector relaxation of the quadratic program. Replace y_i by a vector $V_i \in R^n$
We get,

$$max \sum_{\{i,j\} \in E} W_{ij} \frac{(1 - V_i V_j)}{2}$$

such that $V_i V_j = 1$
This is a vector program which is a relaxation of the quadratic program above because for any feasible solution $(y_1, y_2, y_3, ..., y_n)$ of the quadratic program, the solution $V_i = (y_i, 0, 0, ..., 0) \forall i = 1, 2, 3, ..., n$ is a feasible solution of the vector program. Hence if OPT_V is the cost of an optimal solution of the vector program then $OPT_V \geq OPT$

An important observation by Goemans and Williamson is that this vector program is equivalent to Semi-Definite programming,and SDP can be solved in polynomial time. For this lecture we take this as given,the proof will be given in the next lecture

Note any solution to the vector program(VP) is collection of n vectors each on the surface of the n-dimensional unit sphere S_{n-1}
Also note that

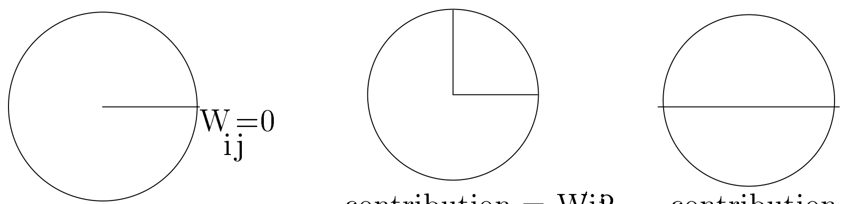
$$V_i V_j = |V_i||V_j| cos(\theta_{ij})$$

Since $|V_i| = |V_j|$, we have $V_i V_j = cos(\theta_{ij})$

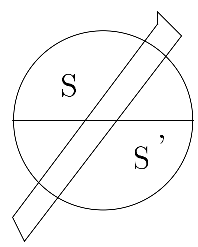
$$OPT_V = \frac{1}{2} \sum_{\{i,j\} \in E} W_{ij} (1 - cos(\theta_{ij}))$$

We can observe that the larger the angle, the more the relaxation says that vertces should lie on the opposite sides. So, we apply a key idea here here of picking a random hyperplane passing through the origin
This is done by picking a unit vector r uniformly at random , uniformly from among all unit vectors in R^n.Define the random hyperplane

$$H = \{X | r.X = 0\}$$



contribution = 0 contribution = Wij/2 contribution = Wij



$$S = \{i | r.V_i \geq 0\}$$

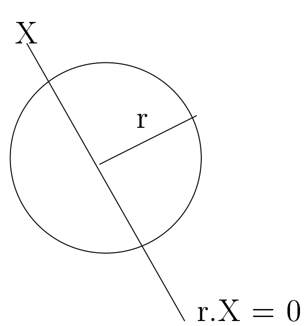
$$\bar{S} = \{i | r.V_i < 0\}$$

Procedure:
Pick $x_1, x_2, x_3, ..., x_n$ from the unit mormal distribution

$$f(x) = \frac{1}{\sqrt(2\pi)} e^{-\frac{x^2}{2}}$$

define

$$d = \frac{(\sum_{1=1}^{n} {x_i}^2)}{2}$$



define $r = (\frac{x_1}{d}, \frac{x_2}{d}, ..., \frac{x_n}{d})$
$r = (y_1, y_2, y_3, ..., y_n)$
probability distribution function for r is given by

$$\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

$$= (\frac{1}{\sqrt{2\pi}})^n e^{-\frac{1}{2}\sum_{i=1}^{n} {y_i}^2}$$

$$= \frac{1}{\sqrt{(2\pi)}} e^{-\frac{1}{2}}$$

which doesnt depend on y_is and is therefore uniform

The probability that for any edge $\{u, v\}$ that $\{u, v\}$ crosses $barS$ can be given as $\frac{\theta_{ij}}{\pi}$. Expected contribution of edge $\{i, j\}$ is $W_{ij}\theta_{ij}$. Total expected cost of the solution is

$$\frac{1}{\pi} \sum_{\{i,j\} \in E} W_{ij}\theta_{ij}$$

We are comparing this with

$$OPT_V = \frac{1}{2} \sum_{\{i,j\} \in E} W_{ij}(1 - cos(\theta_{ij}))$$

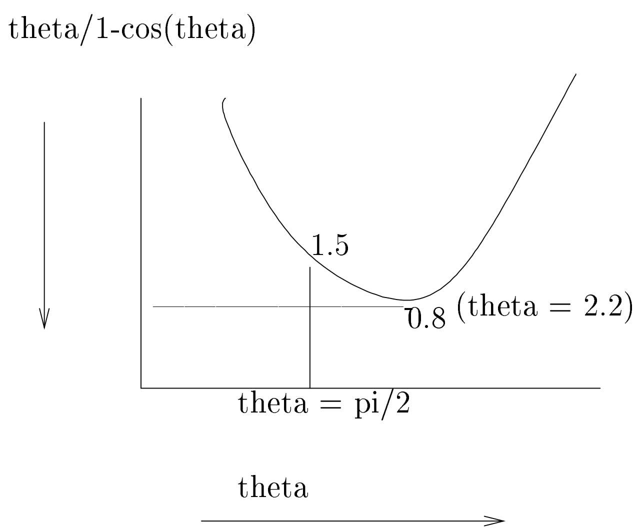
We would like to show that , $\frac{\theta_{ij}}{\pi}$ is not too small as compared to $\frac{(1-cos\theta_{ij})}{2}$. So we look at

$$min \frac{\frac{\theta_{ij}}{\pi}}{\frac{(1-cos\theta_{ij})}{2}}$$

=

$$\frac{2}{\pi} \min_{0 \le \theta_{ij} \le \pi} (\frac{\theta}{1 - cos\theta}) = \alpha$$

where α is our approximation factor ,$\alpha = 0.8785$
which occurs for $\theta = 2.2$.



theta/1-cos(theta)
1.5
0.8
(theta = 2.2)
theta = pi/2
theta