Lecture 6

Lecturer: David P. Williamson *Scribe: Xiaowei Yang*

1 Semidefinite Programming

1.1 MAX CUT, continued

In the last lecture we saw that a *semidefinite program* (SDP)

$$\begin{aligned} \text{Max} \quad & \sum_{i,j} c_{ij} x_{ij} \\ \text{subject to:} \quad & \\ & \sum_{i,j} a_{ijk} x_{ij} = b_k & \forall k \\ & X = (x_{ij}) \succeq 0 & X \text{ is symmetric and } X \in \Re^{n \times n} \end{aligned}$$

is equivalent to a *vector program* (VP) which can be formally stated as

$$\begin{aligned} \text{Max} \quad & \sum_{i,j} c_{ij} (\vec{v_i} \cdot \vec{v_j}) \\ \text{subject to:} \quad & \\ & \sum_{i,j} a_{ijk} (\vec{v_i} \cdot \vec{v_j}) = b_k & \forall k \\ & \vec{v_i} \in \Re^n & i = 1, \cdots, n \end{aligned}$$

where n is the number of vectors in the vector program. This follows since X is symmetric and psd iff $X = V^T V$ i.e., iff $x_{ij} = \vec{v_i} \cdot \vec{v_j}$ where $\vec{v_i}$ is the ith column vector of V. That is,

$$V = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \vec{v_1} & \vec{v_2} & \cdots & \vec{v_n} \\ \vdots & \vdots & & \vdots \end{pmatrix}$$

Last time, we showed that the MAX CUT could be relaxed to the following vector program

$$\begin{aligned} \text{Max} \quad & \frac{1}{2} \sum_{i<j} w_{ij} (1 - \vec{v_i} \cdot \vec{v_j}) \\ & \vec{v_i} \cdot \vec{v_i} = 1 & \forall i \\ & \vec{v_i} \in \Re^n & i = 1, \cdots, n \end{aligned}$$

We considered the **VectorRound** algorithm. Suppose the solution to the vector relaxation program is $\vec{v}$. Draw vector $\vec{r}$ uniformly from unit sphere. $\forall i$,

$$\begin{aligned} i \in S &\quad \text{if } \vec{v_i} \cdot \vec{r} \geq 0 \\ i \notin S &\quad \text{if } \vec{v_i} \cdot \vec{r} < 0 \end{aligned}$$

We proved the following two lemmas:

Lemma 1 $\Pr[i \in S, j \notin S \;\; or \;\; i \notin S, j \in S] = \frac{1}{\pi} \arccos(\vec{v_i} \cdot \vec{v_j})$.

Lemma 2

$$\min_{-1 \leq x \leq 1} \frac{\frac{1}{\pi} \arccos(x)}{\frac{1}{2}(1-x)} \geq .878$$

From these two lemmas, we proved the following theorem:

Theorem 3 *VectorRound is a 0.878-approximation algorithm.*

We proved the theorem by showing:

$$\begin{aligned}
(1) \qquad E[w(S)] &= \sum_{i<j} w_{ij} \frac{1}{\pi} \arccos(\vec{v_i} \cdot \vec{v_j}) \\
(2) \qquad &\geq 0.878 \cdot \frac{1}{2} \sum_{i<j} w_{ij} (1 - \vec{v_i} \cdot \vec{v_j}) \\
&\geq 0.878 \cdot OPT
\end{aligned}$$

(1) is derived from Lemma 1 and (2) is derived from Lemma 2.

Research question: Can you get a .878-approximation algorithm without solving an SDP? In the case of the set cover problem, we had an algorithm that needed to solve an LP, and we were able to show that we could obtain the same performance guarantee without actually solving the LP. Can the same be done here?

1.2 Quadratic Programming

We now show that we can get an approximation algorithm for some kinds of quadratic programming by using the same techniques. Consider quadratic programs of the form

$$\begin{aligned} \text{Max} \quad & \sum_{i,j} a_{ij} (x_i \cdot x_j) \\ & x_i \in \{-1, +1\} \qquad\qquad i = 1, \cdots, n \end{aligned}$$

We get into some conceptual trouble if $OPT < 0$, since then our notion of coming within a factor of $\alpha < 1$ for a maximization problem implies that we would be doing better than optimal! For example, if the $OPT = -12$, a 0.8-approximation algorithm will actually do better — $0.8 \times (-12) > OPT$! We could redefine the notion of an approximation algorithm for this case, but for now we restrict our attention to a case in which the objective function is non-negative. Assume $A = (a_{ij}) \succeq 0$ and symmetric. This solves the problem since the objective function is $x^T Ax$, and we know that since A is psd, then $x^T Ax \geq 0$.

As before we can relax this to the following vector program:

$$
\begin{aligned}
Z_{SDP} \quad = \quad \text{Max} \quad & \sum_{i,j} a_{ij}(\vec{v_i} \cdot \vec{v_j}) & \\
& \vec{v_i} \cdot \vec{v_j} = 1 & \forall i, j \\
& \vec{v_i} \in \Re^n & i = 1, \cdots, n
\end{aligned}
$$

We have $Z_{SDP} \geq OPT$.

Consider the same vector rounding algorithm.

VectorRound2

> Solve the SDP and get vectors $\vec{v}$
> Choose a random vector $\vec{r}$ uniformly from the unit n-sphere
> for $i \leftarrow 1$ to n
>   if $\vec{v_i} \cdot \vec{r} \geq 0$
> $\hat{x}_i \leftarrow 1$
>   else
> $\hat{x}_i \leftarrow -1$

We will now give two lemmas similar to Lemmas 1 and 2.

Lemma 4 $E[\hat{x}_i \cdot \hat{x}_j] = \frac{2}{\pi} \arcsin(\vec{v_i} \cdot \vec{v_j})$

Proof:

$$
\begin{aligned}
E[\hat{x}_i \cdot \hat{x}_j] &= \Pr[\hat{x}_i \cdot \hat{x}_j = 1] - \Pr[\hat{x}_i \cdot \hat{x}_j = -1] \\
&= \left(1 - \frac{1}{\pi} \arccos(\vec{v_i} \cdot \vec{v_j})\right) - \left(\frac{1}{\pi} \arccos(\vec{v_i} \cdot \vec{v_j})\right) \\
&= 1 - \frac{2}{\pi} \arccos(\vec{v_i} \cdot \vec{v_j}) \\
&= 1 - \frac{2}{\pi} \left[\frac{\pi}{2} - \arcsin(\vec{v_i} \cdot \vec{v_j})\right] \\
&= \frac{2}{\pi} \arcsin(\vec{v_i} \cdot \vec{v_j})
\end{aligned}
$$

The second term in the second equality follows from Lemma 1, and the fourth equality follows since $\arcsin(x) + \arccos(x) = \frac{\pi}{2}$. □

We would like our proof to proceed as follows. We would like to prove an equivalent of Lemma 2; suppose there is some α such that the following is true:

Lemma 5

$$\min_{-1\leq x\leq 1} \frac{\frac{2}{\pi}\arcsin(x)}{x} \geq \alpha.$$

We would then like the proof to go the same as before:

$$\begin{aligned}
E[\sum_{i,j} a_{ij}(\hat{x}_i\cdot\hat{x}_j)] &= \sum_{i,j} a_{ij} E[\hat{x}_i\cdot\hat{x}_j] \\
&= \sum_{i,j} a_{ij}\frac{2}{\pi}\arcsin(\vec{v_i}\cdot\vec{v_j}) \\
&\geq \alpha\sum_{i,j} a_{ij}(\vec{v_i}\cdot\vec{v_j}) \qquad (3)\\
&\geq \alpha\cdot OPT.
\end{aligned}$$

But we cannot do this because the inequality (3) is not correct. This is because some of the a_{ij}'s may be negative. The inequality $\frac{2}{\pi}\arcsin(x)\geq \alpha x$ will become $\frac{2}{\pi}a_{ij}\arcsin(x)\leq \alpha x a_{ij}$ if $a_{ij}<0$.

Thus to analyze this algorithm, we will have to do something different. We will have to consider comparing the expected value of the solution to the SDP value on a global basis, rather than a term-by-term basis. We introduce some facts that we state without proof.

Fact 1 *If $A\succeq 0$, $B\succeq 0$, then $\sum_{ij} a_{ij}b_{ij}\geq 0$.*

Fact 1 is easy to prove.

Fact 2 *If $X\succeq 0$, $|x_{ij}|\leq 1$, $\forall i,j$ and $Z=(z_{ij})$ such that $z_{ij}=\arcsin(x_{ij})-x_{ij}$, then $Z\succeq 0$.*

Fact 2 is harder to prove. The proof uses Taylor expansion of matrix Z.

Theorem 6 *(Nesterov '97) VectorRound2 is a $\frac{2}{\pi}$-approximation algorithm.*

Proof: We want to show

$$E[\sum_{ij} a_{ij}(\hat{x}_i\cdot\hat{x}_j)] \geq \frac{2}{\pi}\sum_{ij} a_{ij}(\vec{v_i}\cdot\vec{v_j}) \geq \frac{2}{\pi}\cdot OPT$$

We know

$$E[\sum_{ij} a_{ij}(\hat{x}_i\cdot\hat{x}_j)] = \frac{2}{\pi}\sum_{ij} a_{ij}\arcsin(\vec{v_i}\cdot\vec{v_j})$$

So we want to show

$$\frac{2}{\pi}\sum_{ij} a_{ij}\arcsin(\vec{v_i}\cdot\vec{v_j}) - \frac{2}{\pi}\sum_{ij} a_{ij}(\vec{v_i}\cdot\vec{v_j}) \geq 0$$

which is

$$\frac{2}{\pi}\sum_{ij} a_{ij}(\arcsin(\vec{v_i}\cdot\vec{v_j}) - (\vec{v_i}\cdot\vec{v_j})) \geq 0$$

Setting $x_{ij} = \vec{v_i}\cdot\vec{v_j}$, we obtain that $X = (x_{ij}) \succeq 0$, symmetric, and $|x_{ij}| \leq 1$, $\forall i, j$, since

$$-\|\vec{v_i}\cdot\vec{v_j}\| \leq x_{ij} = \vec{v_i}\cdot\vec{v_j} \leq \|\vec{v_i}\|\|\vec{v_j}\|$$

As $v_i, \forall i$ is a unit vector, $\|\vec{v_i}\cdot\vec{v_j}\| \leq 1$. Thus the left-hand side is equal to

$$\frac{2}{\pi}\sum_{ij} a_{ij}(\arcsin(x_{ij}) - x_{ij}).$$

Setting $z_{ij} = \arcsin(x_{ij}) - x_{ij}$, this is equal to

$$\frac{2}{\pi}\sum_{ij} a_{ij}z_{ij} \geq 0,$$

since $Z = (z_{ij})$ is psd by Fact 2 and thus $\sum_{ij} a_{ij}z_{ij} \geq 0$ by Fact 1. Hence the result. $\square$

We have seen several techniques used to design an approximation algorithm, which include:

- Rounding, which is used to obtain an approximation for an integer problem from LP.
- Dual Rounding
- Randomized Rounding
- Primal-Dual method
- Dynamic programming
- Randomization

2 Scheduling problems and LP

2.1 Some deterministic scheduling notation

We have seen several techniques used to design an approximation algorithm, which include:

- Rounding, which is used to obtain an approximation for an integer problem from LP.

- Dual Rounding
- Randomized Rounding
- Primal-Dual method
- Dynamic programming
- Randomization

We now turn to the use of some of these techniques to obtain approximation algorithms for deterministic scheduling problems. Before we get to that, we will need to review some scheduling notation due to Graham, Lawler, Lenstra, and Rinnooy Kan. In their notation, scheduling problems are represented as *Machine environment*|*Constraints*|*Objective function*. We give some examples of the possibilities. Two possible elements in the environment field are 1 (for single machine problems) and P (for identical parallel machine problems). Two possible elements in the constraint field are r_j (when each job j has a "release date". Then job j is not available before time r_j) and $prec$ (when there are some precedence constraints, e.g. $j \prec k$. This implies that we can not start processing job k before job j is finished). Assume C_j is the completion time of job j. Then two possible elements in the objective field are C_{max} (meaning $\min \max_j C_j$, the maximum completion time of all jobs) and $\sum_j w_j C_j$ (meaning $\min \sum_j w_j C_j$, where w_j is weight of job j).

2.2 $1||\sum_j w_j C_j$

We now turn to the problem $1||\sum_j w_j C_j$.

- **Input:** n jobs $J_1 ... J_n$ with weights $w_1 ... w_n$ and processing times $p_1 ... p_n$.
- **Goal:** Find a schedule on 1 machine that minimizes $\sum_j w_j C_j$.

This problem is polynomial time solvable. But even minor twists make the problem hard: $1|r_j|\sum_j w_j C_j$ and $1|prec|\sum_j w_j C_j$ are NP-hard.

2.3 $1|prec|\sum_j w_j C_j$

To obtain an approximation algorithm, we want to develop a linear programming relaxation for the scheduling problem. To get one, let us think in terms of "mean busy time"; that is, the average of all the points in time at which job j executes. Let M_j denote the mean busy time of job j, which is defined as

$$M_j = \frac{\int_t I_j(t) t dt}{\int_t I_j(t) dt}$$

$I_j(t)$ is defined as follows:

$$I_j(t) = \begin{cases} 1 & \text{if job } j \text{ is processed at time } t \\ 0 & \text{o.w.} \end{cases}$$

For a single job j,

$$\begin{aligned} M_j &= \frac{\int_t I_j(t)t \cdot dt}{\int_t I_j(t)dt} \\ &= \frac{\int_{C_j-p_j}^{C_j} t \cdot dt}{\int_{C_j-p_j}^{C_j} dt} \\ &= \frac{\frac{1}{2}t^2|_{C_j-p_j}^{C_j}}{p_j} \\ &= \frac{\frac{1}{2}(C_j^2 - (C_j - p_j)^2)}{p_j} \\ &= \frac{\frac{1}{2}(2p_jC_j - p_j^2)}{p_j} \\ &= C_j - \frac{p_j}{2} \end{aligned}$$

Observe that this is the midpoint of the completion time of job j, halfway inbetween its start time and its completion time.

Now, suppose $S \subseteq \{1, 2, \ldots, n\}$ is a set of jobs. We introduce the following notation:

$$\begin{aligned} M_S &\equiv \text{mean busy time of a set of jobs } S \\ p(S) &= \sum_{j \in S} p_j \\ I_S(t) &= \sum_{j \in S} I_j(t) \end{aligned}$$

M_S is the mean busy time of the set of jobs in S; that is, the average point in time at which a job in S is executed. M_S can be calculated as:

$$\begin{aligned}
M_S &= \frac{\int_t I_S(t)t\,dt}{\int_t I_S(t)dt} \\
&= \frac{\int_t(\sum_{j\in S} I_j(t))t\,dt}{p(S)} \\
&= \frac{\sum_{j\in S}\int_t I_j(t)t\,dt}{p(S)} \\
&= \frac{\sum_{j\in S}(p_jC_j - \frac{1}{2}p_j^2)}{p(S)} \\
&= \frac{\sum_{j\in S} p_jM_j}{p(S)}
\end{aligned}$$

As some students pointed out in class, M_S is the weighted average of $M_j, \forall j \in S$.

It is not hard to see that for any set S, $M_S \geq \frac{1}{2}p(S)$: the earliest point in time at which jobs in S can begin executing is time 0, and thus the earliest point at which they can finish is time $p(S)$. Hence the lowest possible value of M_S is $\frac{1}{2}p(S)$, which is achieved when jobs in S are all scheduled first.

We can now use the above write a linear programming relaxation of the scheduling problem $1|prec|\sum_j w_jC_j$ in terms of variables M_j:

$$\text{Min} \quad \sum_{j=1}^{n} w_j(M_j + \frac{p_j}{2})$$

subject to:

$$\sum_{j\in S} p_jM_j \geq \frac{1}{2}p^2(S) \qquad \forall S \subseteq \{1,\ldots,n\}$$

$$M_j + \frac{p_j}{2} \leq M_k - \frac{p_k}{2} \qquad \forall j \prec k$$

We claim without proof that this LP is solvable in polynomial time. Now, given the solution, how can we obtain a schedule? We do this via a form of deterministic rounding. Suppose $\bar{M}$ is the optimal solution and

$$\bar{M}_1 \leq \bar{M}_2 \leq \bar{M}_3 \leq \cdots \leq \bar{M}_n$$

Now consider the following algorithm:

Schedule-by-$\bar{M}_j$

Schedule jobs in the same order as the values of $\bar{M}_j$.

Observe that gives a feasible solution: if j must precede k because of a precedence constraint, it must be the case that $\bar{M}_j < \bar{M}_k$ by the LP constraint. Thus the algorithm schedules job j before job k.

We can prove the following lemma about the LP solution.

Lemma 7 $\bar{M}_j \geq \frac{1}{2}(\sum_{k=1}^{j} p_k)$

Proof: For $S = \{1, 2, \ldots, j\}$, from the LP constraint, we know that

$$\sum_{k=1}^{j} p_k \bar{M}_k \geq \frac{1}{2}\left(\sum_{k=1}^{j} p_k\right)^2.$$

Since $\bar{M}_j \geq \bar{M}_k$ for $k \leq j$, we have

$$\bar{M}_j \sum_{k=1}^{j} p_k \geq \sum_{k=1}^{j} p_k \bar{M}_k \geq \frac{1}{2}\left(\sum_{k=1}^{j} p_k\right)^2.$$

Dividing both sides by $\sum_{k=1}^{j} p_k$ on both sides gives the lemma statement. □

The lemma leads to the following theorem.

Theorem 8 *(Hall, Schulz, Shmoys, Wein '96) Schedule-by-$\bar{M}_j$ is a 2-approximation algorithm for $1|prec|\sum_j w_j C_j$.*

Proof: Let $\hat{C}_j$ denote the completion time of job j in the schedule we construct. Then for each job j,

$$\hat{C}_j = \sum_{k=1}^{j} p_k \leq 2\bar{M}_j.$$

It follows that

$$\begin{aligned}
\sum_{j=1}^{n} w_j \hat{C}_j &\leq 2\sum_{j=1}^{n} w_j \bar{M}_j \\
&\leq 2\sum_{j=1}^{n} w_j (\bar{M}_j + \frac{p_j}{2}) \\
&= 2Z_{LP} \\
&\leq 2OPT.
\end{aligned}$$

□

2.4 $1|r_j|\sum_j w_j C_j$

We finally turn to the problem $1|r_j|\sum_j w_j C_j$. In order to get an approximation algorithm for this problem, we consider a different linear programming relaxation of the problem. We introduce variables y_{jt} where

$$y_{jt} = \begin{cases} 1 & \text{if job } j \text{ processed in time } [t, t+1) \\ 0 & \text{otherwise} \end{cases}$$

Let T denote the max possible time any job can execute, which is at most

$$\max_j r_j + \sum_{j=1}^{n} p_j$$

We now formulate a series of constraints for the LP. First note that

$$\sum_{j=1}^{n} y_{jt} \leq 1, \text{ for } t = 0, \ldots, T$$

since at most one job can be processed at any point in time. In order to impose that no job is processed before its release date, we have

$$y_{jt} = 0 \quad \forall t = 0, \ldots, r_j - 1 \quad \forall j$$

Also,

$$\sum_{t=r_j}^{T} y_{jt} = p_j \quad j = 1, \cdots, n$$

since the total amount of processing for job j must be p_j.

Finally, we compute M_j in terms of the variables y_{jt}.

$$\begin{aligned} M_j &= \frac{1}{p_j} \int_{r_j}^{T} I_j(t) \cdot t \cdot dt \\ &= \frac{1}{p_j} \sum_{t=r_j}^{T} y_{jt} \frac{1}{2} \tau^2 |_{\tau=t}^{\tau=t+1} \\ &= \frac{1}{p_j} \sum_{t=r_j}^{T} y_{jt} \left(\frac{1}{2}(t+1)^2 - \frac{1}{2} t^2 \right) \\ &= \frac{1}{p_j} \sum_{t=r_j}^{T} y_{jt} \left(t + \frac{1}{2} \right) \end{aligned}$$

Thus we have the following linear programming relaxation of the problem $1|r_j|\sum_j w_j C_j$:

$$\text{Min} \qquad \sum_{j=1}^{n} w_j \left(M_j + \frac{p_j}{2} \right)$$

subject to:

$$\begin{aligned} & M_j = \frac{1}{p_j} \sum_{t=r_j}^{T} y_{jt} (t + \frac{1}{2}) && j = 1, \cdots, n \\ & \sum_{j=1}^{n} y_{jt} \leq 1 && t = 0, \ldots, T \\ & y_{jt} = 0 && t = 0, \ldots, r_j - 1, \forall j \\ & \sum_{t=r_j}^{T} y_{jt} = p_j && j = 1, \cdots, n. \end{aligned}$$

Oddly enough, it is possible to show that if the p_j are integral, then an optimal extreme point has y_{jt} that are either 0 or 1; that is, the solution is integral. It has usually been the case that an integral solution is good for us; it implies that we achieve the optimal solution. However, in this case, the value $M_j + p_j/2$ is only equal to the completion time of a job if all p_j units of job j are scheduled consecutively; otherwise $M_j + p_j/2$ underestimates the completion time. Thus if Z_{LP} is the value of this LP relaxation, then $Z_{LP} \leq OPT$.

This linear program seems very bad because we have an exponential number of variables (in the size of the input) and an exponential number of constraints. But it turns out this is not so bad.

Claim 9 *1. The LP can be solved in polynomial-time;*

2. An optimal solution can be written down in polynomial space by recording intervals during which each job is executed.

In fact, the following claim tells us how to obtain an optimal solution.

Claim 10 *The following gives an optimal solution to LP. Assume*

$$\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \cdots \geq \frac{w_n}{p_n}.$$

Then schedule the available job of the smallest index.

However, it does not directly give us a solution for $1|r_j|\sum_j w_j C_j$, because the LP solution may have jobs that are preempted. Preemptions occur due to release dates of higher priority jobs; thus at most n preemptions occur for this reason, and at most $O(n)$ time intervals are needed to describe an optimal solution.

To get an approximation algorithm given an LP solution, we invoke our old friend, randomized rounding.

RandomRound

Solve LP, get optimal solution (y^*, M^*).
Set $T_j = t + \frac{1}{2}$ with probability $\frac{y^*_{jt}}{p_j}$ $, \forall j$
Schedule jobs as early as possible in same order as values of T_j

Suppose that $T_1 \leq T_2 \leq \cdots \leq T_n$. Let $\hat{C}_j$ be the random variable giving the completion time of job j. We begin the analysis by considering the expected value of $\hat{C}_j$ given a fixed value of T_j.

Lemma 11 $E[\hat{C}_j | T_j = \tau] \leq p_j + 2\tau$

Proof:

Let I be a random variable giving the total idle time before job j is processed, and let P be a random variable giving total amount of time spent processing jobs before j starts. Then obviously

$$\hat{C}_j = I + P + p_j.$$

Since the idle time before job j can only result from the release dates of jobs to be scheduled before job j, the idle time is at worst

$$\max_{k:T_k \leq T_j} r_k \leq \max_{k:T_k \leq T_j} T_k \leq T_j = \tau$$

so that $I \leq \tau$.

Furthermore,

$$\begin{aligned}
E[P|T_j = \tau] &= \sum_{k \neq j} p_k \Pr[\text{job } k \text{ is processed before } j | T_j = \tau] \\
&= \sum_{k \neq j} p_k \Pr[T_k \leq T_j | T_j = \tau] \\
&= \sum_{k \neq j} p_k \left(\sum_{t=0}^{\tau-1} \frac{y^*_{kt}}{p_k} \right) \\
&= \sum_{k \neq j} \sum_{t=0}^{\tau-1} y^*_{kt} \\
&= \sum_{t=0}^{\tau-1} \sum_{k \neq j} y^*_{kt} \\
&\leq \tau,
\end{aligned}$$

since $\sum_{k \neq j} y^*_{kt} \leq 1$ by the linear program.

Therefore

$$E[\hat{C}_j | T_j = \tau] = p_j + E[I|T_j = \tau] + E[P|T_j = \tau] \leq p_j + 2\tau.$$

$\square$

We can now prove the following theorem.

Theorem 12 *(Schulz, Skutella '96) RandomRound is a 2-approximation algorithm for* $1|r_j|\sum_j w_j C_j$

Proof: Notice that

$$\begin{aligned}
E[\hat{C}_j] &\leq p_j + 2 \sum_{t=0}^{T} \left(t + \frac{1}{2} \right) \Pr\left[T_j = t + \frac{1}{2} \right] \\
&= p_j + 2 \sum_{t=0}^{T} \left(t + \frac{1}{2} \right) \frac{y^*_{jt}}{p_j} \\
&= p_j + 2M^*_j \qquad (4) \\
&= 2 \left(M^*_j + \frac{p_j}{2} \right).
\end{aligned}$$

Inequality (4) follows from the LP formulation. Thus

$$E[\sum_j w_j \hat{C}_j] = \sum_j w_j E[\hat{C}_j] \leq 2 \sum_j w_j (M_j^* + \frac{p_j}{2}) \leq 2OPT$$

$\square$

Here is a historical review of approximation algorithms for this problem.

The first 2-approximation algorithm for this problem was due to Goemans. Here's an alternate perspective on what the algorithm is doing: the LP solution is a preemptive schedule (one in which the processing of any job j need not be continuous). Pick $\alpha_j \in [0,1]$ uniformly for each j. Let T_j be α_j-point of job j: that is, the time when $\alpha_j p_j$ units of job j have been processed in the preemptive schedule. Then schedule jobs according to T_j as before. The $O(n \log n)$-time algorithm of Goemans to find the LP solution actually finds this preemptive schedule.

Goemans, Queyranne, Schulz, Skutella, and Wang (1998) gave a 1.6853-approximation algorithm by picking $\alpha_j \in [0,1]$ non-uniformly.

In 1999, a PTAS for the problem was given by 5 different research groups independently.