4 Deterministic rounding of linear programs

- An integer programming (IP) formulation of a problem can be relaxed it to a linear program (LP)
- We solve the LP to obtain a fractional solution, then round it to an integer solution
- The easiest way to round the fractional solution to an integer solution in which all values are 0 or 1 is to take variables with relatively large values and round them up to 1, while rounding all other variables down to 0

4.1 Minimizing the sum of completion times on a single machine

- Consider the problem of scheduling jobs on a single machine so as to minimize the sum of the job completion times
- In particular, we are given as input $n$ jobs, each of which has a processing time $p_j$ and release date $r_j$
- The values $p_j$ and $r_j$ are integers such that $r_j \geq 0$ and $p_j > 0$
Construct a schedule for these jobs such that
– at most one job is processed at each point in time,
– no job is processed before its release date, and
– each job must be processed nonpreemptively

That is, once a job begins to be processed, it must be processed completely before any other job begins its processing.

If \( C_j \) denotes the time at which job \( j \) is finished processing, then the goal is to find the schedule that minimizes \( \sum_{j=1}^{n} C_j \).

This objective is equivalent to minimizing the average completion time, since the average completion time just rescales the objective function for each feasible solution by a factor of \( 1/n \).

Below we will show that we can convert any preemptive schedule into a nonpreemptive schedule in such a way that the completion time of each job at most doubles.
In a preemptive schedule, we can still schedule only one job at a time on the machine, but we do not need to complete each job’s required processing consecutively— we can interrupt the processing of a job with the processing of other jobs.

In the preemptive version of this problem, the goal is to find a preemptive schedule that minimizes the sum of the completion times.

An optimal solution to the preemptive version of the scheduling problem can be found in polynomial time via the shortest remaining processing time (SRPT) rule, which is as follows:

- We start at time 0 and schedule the job with the smallest amount of remaining processing time as long as the job is past its release date and we have not already completed it.
- We schedule it until either it is completed or a new job is released.
• Let $C_j^p$ be the completion time of job $j$ in an optimal preemptive schedule, and let $OPT$ be the sum of completion times in an optimal nonpreemptive schedule.

• We have the following observation

Observation 4.1:

$$\sum_{j=1}^{n} C_j^p \leq OPT.$$ 

Proof: This immediately follows from the fact that an optimal nonpreemptive schedule is feasible for the preemptive scheduling problem.

Consider the following scheduling algorithm

• Find an optimal preemptive schedule by SRPT

• We schedule the jobs nonpreemptively in the same order that they complete in this schedule

• To be more precise, suppose that the jobs are indexed such that $C_1^p \leq C_2^p \leq \cdots \leq C_n^p$

• Then we schedule job 1 from its release date $r_1$ to time $r_1 + p_1$

• We schedule job 2 to start as soon as possible after job 1; that is, we schedule it from $\max(r_1 + p_1, r_2)$ to $\max(r_1 + p_1, r_2) + p_2$

• The remaining jobs are scheduled analogously.
If we let $C_j^N$ denote the completion time of job $j$ in the nonpreemptive schedule that we construct, for $j = 1, \ldots, n$, then job $j$ is processed from $\max\{C_{j-1}^N, r_j\}$ to $\max\{C_{j-1}^N, r_j\} + p_j$.

Scheduling nonpreemptively in this way does not delay the jobs by too much.

**Lemma 4.2:** For each job $j = 1, \ldots, n$

$$C_j^N \leq 2C_j^P.$$

**Proof:** Let us first derive a couple of easy lower bounds on $C_j^P$. Since we know that $j$ is processed in the optimal preemptive schedule after jobs $1, \ldots, j - 1$, we have

$$C_j^P \geq \max_{k=1,\ldots,j} r_k \quad \text{and} \quad C_j^P \geq \sum_{k=1}^j p_k.$$

By construction it is also the case that

$$C_j^N \geq \max_{k=1,\ldots,j} r_k.$$
Consider the nonpreemptive schedule constructed by the algorithm, and focus on any period of time that the machine is idle. Idle time occurs only when the next job to be processed has not yet been released. Consequently, in the time interval from $\max_{k=1,...,j} r_k$ to $C_j^N$, there cannot be any point in time at which the machine is idle. Therefore, this interval can be of length at most $\sum_{k=1}^{j} p_k$ since otherwise we would run out of jobs to process. This implies that

$$C_j^N \leq \max_{k=1,...,j} r_k + \sum_{k=1}^{j} p_k \leq 2C_j^P,$$

where the last inequality follows from the two lower bounds on $C_j^P$ derived above.

**Theorem 4.3:** Scheduling in order of the completion times of an optimal preemptive schedule is a 2-approximation algorithm for scheduling a single machine with release dates to minimize the sum of completion times.

**Proof:** We have that

$$\sum_{j=1}^{n} C_j^N \leq 2 \sum_{j=1}^{n} C_j^P \leq 2 \text{OPT},$$

where the first inequality follows by Lemma 4.2 and the second by Observation 4.1.
4.2 Minimizing the weighted sum of completion times on a single machine

- Consider a generalization of the problem in which each job has a weight \( w_j \geq 0 \), and our goal is to minimize the weighted sum of completion times.
- If \( C_j \) denotes the time at which job \( j \) is finished processing, then the goal is to find a schedule that minimizes \( \sum_{j=1}^{n} w_j C_j \).
- The previous problem is the unweighted and the problem in this section the weighted case.

Unlike the unweighted case, it is NP-hard to find an optimal schedule for the preemptive version of the weighted case.

The previous section gives us a way to round any preemptive schedule to one whose sum of weighted completion times is at most twice more.

We cannot, though, use the same technique of finding a lower bound on the cost of the optimal nonpreemptive schedule by finding an optimal preemptive schedule.
We can still get a constant approximation algorithm for this problem by using some of the ideas from the previous section.

To obtain the 2-approximation algorithm in the previous section, we used that $C_j^N \leq 2C_j^P$.

If we look at the proof of Lemma 4.2, to prove this inequality we used only that the completion times $C_j^P$ satisfied $C_j^P \geq \max_{k=1,...,j} r_k$ and $C_j^P \geq \sum_{k=1}^j P_k$ (assuming that jobs are indexed such that $C_1^P \leq C_2^P \leq \cdots \leq C_n^P$).

Furthermore, in order to obtain an approximation algorithm, we needed that $\sum_{j=1}^n C_j^P \leq \text{OPT}$.

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$. 

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$. 

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$. 

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$. 

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$. 

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$. 

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$. 

We can give a LP relaxation of the problem with variables $C_j$ such that these inequalities hold within a constant factor, which in turn will lead to a constant factor approximation algorithm for the problem.

To construct our LP relaxation, we will let the variable $C_j$ denote the completion time of job $j$.

Then our objective function is clear: we want to minimize $\sum_{j=1}^n w_j C_j$.

The first set of constraints is easy: for each job $j = 1,...,n$, job $j$ cannot complete before it is released and processed, so that $C_j \geq r_j + p_j$.
To introduce the second set of constraints, consider some set \( S \subseteq \{1, \ldots, n\} \) of jobs.

Consider the sum \( \sigma = \sum_{j \in S} p_j c_j \).

It is minimized when all the jobs in \( S \) have a release date of zero and all the jobs in \( S \) finish first in the schedule.

Then any completion time \( c_j \) for \( j \in S \) is equal to \( p_j \) plus the sum of all the processing times of the jobs in \( S \) that preceded \( j \) in the schedule.

Then in the product \( p_j c_j \), \( p_j \) multiplies itself and the processing times of all jobs in \( S \) preceding \( j \) in the schedule.

The sum \( \sigma \) must contain the term \( p_j p_k \) for all pairs \( j, k \in S \), since either \( k \) precedes \( j \) or \( j \) precedes \( k \) in the schedule.

Simplifying notation, let \( S = \{1, \ldots, n\} \), and let \( p(S) = \sum_{j \in S} p_j \), so that the inequality above becomes \( \sum_{j \in S} p_j c_j \geq \frac{1}{2} p(S)^2 \).

The sum \( \sigma \) can be greater only if the jobs in \( S \) have a release date greater than zero or do not finish first in the schedule, so the inequality must hold unconditionally for any \( S \subseteq N \).
Thus, our second set of constraints is

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2} p(S)^2$$

for each $S \subseteq N$

This gives our LP relaxation for the scheduling problem:

$$\text{minimize } \sum_{j=1}^{n} w_j C_j$$

subject to

$$C_j \geq r_j + p_j, \forall j \in N,$$

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2} p(S)^2, \forall S \subseteq N \quad (4.3)$$

By the arguments above, this LP is a relaxation of the problem, so that for an optimal LP solution $C^*, \sum_{j=1}^{n} w_j C_j^* \leq \text{OPT}$, where OPT denotes the value of the optimal solution to the problem.

There are an exponential number of the second type of constraint, but we will show later that we can use an algorithm called the ellipsoid method to solve this linear program in polynomial time.

Let $C^*$ be an optimal solution for the relaxation that we obtain in polynomial time.
We schedule the jobs nonpreemptively in the same order as of the completion times of $C^*$.

That is, suppose that the jobs are reindexed so that $C_1^* \leq C_2^* \leq \cdots \leq C_n^*$.

Then, as previously, we schedule job 1 from its release date $r_1$ to time $r_1 + p_1$.

We schedule job 2 to start as soon as possible after job 1; that is, we schedule it from $\max(r_1 + p_1, r_2)$ to $\max(r_1 + p_1, r_2) + p_2$.

The remaining jobs are scheduled similarly.

This gives a 3-approximation algorithm for the problem.

**Theorem 4.4:** Scheduling in order of the completion time of $C^*$ is a 3-approximation algorithm for scheduling jobs on a single machine with release dates to minimize the sum of weighted completion times.

**Proof:** Again, assume that the jobs are reindexed so that $C_1^* \leq C_2^* \leq \cdots \leq C_n^*$. Let $C_j^N$ be the completion time of job $j$ in the schedule we construct. We will show that $C_j^N \leq 3C_j^*$ for each $j = 1, \ldots, n$. Then we have that $\sum_{j=1}^{n} w_j C_j^N \leq 3 \sum_{j=1}^{n} w_j C_j^* \leq 3 \text{OPT}$, which gives the desired result.
As in the proof of Lemma 4.2, there cannot be any idle time between \( \max_{k=1,...,j} r_k \) and \( C_j^N \), and therefore it must be the case that
\[
C_j^N \leq \max_{k=1,...,j} r_k + \sum_{k=1}^j p_k .
\]
Let \( \ell \in \{1,...,j\} \) be the index of the job that maximizes \( \max_{k=1,...,j} r_k \) so that
\[
r_\ell = \max_{k=1,...,j} r_k .
\]
By the indexing of the jobs, \( C_j^* \geq C_\ell^* \), and \( C_\ell^* \geq r_\ell \) by the first LP constraint; thus,
\[
C_j^* \geq \max_{k=1,...,j} r_k .
\]
Let \( [j] \) denote the set \( \{1,...,j\} \). We will argue that \( C_j^* \geq \frac{1}{2} p([j]) \), and from these simple facts, it follows that
\[
C_j^N \leq p([j]) + \max_{k=1,...,j} r_k \leq 2C_j^* + C_j^* = 3C_j^* .
\]
Let \( S = [j] \). From the fact that \( C^* \) is a feasible LP solution, we know that
\[
\sum_{k \in S} p_k C_k^* \geq \frac{1}{2} p(S)^2 .
\]
However, by our relabeling, \( C_j^* \geq \cdots \geq C_1^* \), and hence
\[
C_j^* \sum_{k \in S} p_k = C_j^* \cdot p(S) \geq \sum_{k \in S} p_k C_k^* .
\]
By combining these two inequalities and rewriting, we see that \( C_j^* \cdot p(S) \geq \frac{1}{2} p(S)^2 \).
Dividing both sides by \( p(S) \) shows that
\[
C_j^* \geq \frac{1}{2} p(S) = \frac{1}{2} p([j]) .
\]
4.3 Solving large linear programs in polynomial time via the ellipsoid method

- The most popular and practical algorithm for solving LPs is known as the *simplex method*.
- Simplex method is quite fast in practice.
- However, there is no known variant of the algorithm that runs in polynomial time.
- *Interior-point methods* are a class of algorithms for solving linear programs; while typically not as fast or as popular as the simplex method, they do solve linear programs in polynomial time.

However, this isn’t sufficient for solving the linear program above because the size of the linear program is exponential in the size of the input scheduling instance.

Therefore, we will use a linear programming algorithm called the *ellipsoid method*.

Because we will use this technique frequently, we will discuss the general technique before turning to how to solve our particular linear program.
• Suppose we have the following general linear program:

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n} d_j x_j \\
\text{subject to} & \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, n, \\
& \quad x_j \geq 0, \quad \forall j
\end{align*}
\]

• Suppose that we can give a bound $\phi$ on the number of bits needed to encode any inequality $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$

• Then the ellipsoid method for LP allows us to find an optimal solution to the LP in time polynomial in $n$ (the number of variables) and $\phi$, given a polynomial-time separation oracle (which we will define momentarily)

• It is sometimes desirable for us to obtain an optimal solution that has an additional property of being a basic solution; we do not define basic solutions here, but the ellipsoid method will return such basic optimal solutions
• Note that this running time does not depend on $m$, the number of constraints of the LP.

• Thus, as in the case of the previous LP for the scheduling problem, we can solve LPs with exponentially many constraints in poly time given that we have a poly-time separation oracle.

• Such an oracle takes as input a solution $x$ to the LP, and either verifies that $x$ is indeed a feasible solution to the LP or, if it is infeasible, produces a constraint that is violated by $x$.

• If it is not the case that $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ for each $i = 1, \ldots, m$, then the separation oracle returns some constraint $i$ such that $\sum_{j=1}^{n} a_{ij} x_j < b_i$.

• The notes at the end of the chapter, sketch how a polynomial-time separation oracle leads to a polynomial-time algorithm for solving LPs with exponentially many constraints.

• It is truly remarkable that such LPs are efficiently solvable.

• Here, however, efficiency is a relative term: the ellipsoid method is not a practical algorithm.

• For exponentially large LPs that we solve via the ellipsoid method, it is sometimes the case that the LP can be written as a polynomially sized LP, but it is more convenient to discuss the larger LP.
• Even if there is no known way of rewriting the exponentially sized LP one can heuristically find an optimal solution to the LP by repeatedly using any LP algorithm (simplex) on a small subset of the constraints and using the separation oracle to check if the current solution is feasible
• If yes, then we have solved the LP, if not, we add the violated constraints to the LP and resolve
• Practical experience shows that this approach is much more efficient than can be theoretically justified, and should be in the algorithm designer’s toolkit

We now turn to the scheduling problem at hand, and give a polynomial-time separation oracle for the constraints (4.3)
• Given a solution $C$, let us reindex the variables so that $C_1 \leq C_2 \leq \cdots \leq C_n$
• Let $S_1 = \{1\}$, $S_2 = \{1, 2\}$, $\ldots$, $S_n = \{1, \ldots, n\}$
• We claim that it is sufficient to check whether the constraints are violated for the $n$ sets $S_1, \ldots, S_n$
• If any of these $n$ constraints are violated, then we return the set as a violated constraint
• If not, we show below that all constraints are satisfied
Lemma 4.5: Given variables $C_j$, if constraints (4.3) are satisfied for the $n$ sets $S_1, \ldots, S_n$, then they are satisfied for all $S \subseteq N$.

Proof: Let $S$ be a constraint that is not satisfied; that is, $\sum_{j \in S} p_j C_j < \frac{1}{2} p(S)^2$. We will show that then there must be some set $S_i$ that is also not satisfied. We do this by considering changes to $S$ that decrease the difference $\sum_{j \in S} p_j C_j - \frac{1}{2} p(S)^2$. Any such change will result in another set $S'$ that also does not satisfy the constraint.

Note that removing a job $k$ from $S$ decreases this difference if

$$-p_k C_k + p_k p(S - k) + \frac{1}{2} p_k^2 < 0,$$

or if $C_k > p(S - k) + \frac{1}{2} p_k$. Adding a job $k$ to $S$ decreases this difference if

$$p_k C_k - p_k p(S) - \frac{1}{2} p_k^2 < 0,$$

or if $C_k < p(S) + \frac{1}{2} p_k$.

Now let $\ell$ be the highest indexed job in $S$. We remove $\ell$ from $S$ if $C_\ell > p(S - \ell) + \frac{1}{2} p_\ell$. 
By the reasoning above the resulting set $S - \ell$ also does not satisfy the corresponding constraint (4.3). We continue to remove the highest indexed job in the resulting set until finally we have a set $S'$ such that its highest indexed job $\ell$ has $C_{\ell} \leq p(S' - \ell) + \frac{1}{2}p_\ell < p(S')$ (using $p_\ell > 0$). Now suppose $S' \neq S_\ell = \{1, \ldots, \ell\}$. Then there is some $k < \ell$ such that $k \notin S'$. Then since $C_k \leq C_\ell < p(S') < p(S') + \frac{1}{2}p_k$, adding $k$ to $S'$ can only decrease the difference. Thus, we can add all $k < \ell$ to $S'$, and the resulting set $S'_\ell$ will also not satisfy the constraint (4.3).