Linear program

Separation Oracle

Rounding

- We consider a single-machine scheduling problem, and see another way of rounding fractional solutions to integer solutions.
- We will see that by solving a relaxation, we are able to get information on how the jobs might be ordered.
- We construct a solution in which we schedule jobs in the same order as given by the relaxation, and we are able to show that his leads to a good solution .

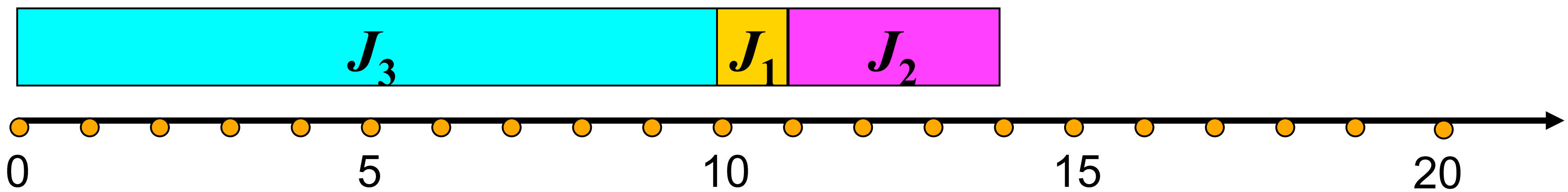
$1|r_j|\Sigma C_j$

- Single machine
- $\boldsymbol{J} = \{1,\ldots, n\}$ – jobs
- $p_j \geq 0$ – processing time of job j.
- $r_j \geq 0$ – release time of job j.
- $C_j(\sigma)$ – completion time of job j in σ.
- No preemption.
- The machine cannot process two jobs at the same time.

Example

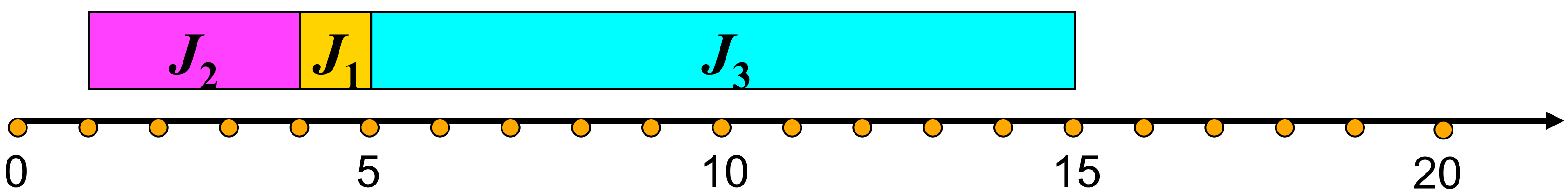
σ_1

$$C_1(\sigma_1) + C_2(\sigma_1) + C_3(\sigma_1) = 35$$



σ_2

$$C_1(\sigma_2) + C_2(\sigma_2) + C_3(\sigma_2) = 24$$



J_1	$p_1 = 1$	$r_1 = 2$
J_2	$p_2 = 3$	$r_2 = 1$
J_3	$p_3 = 10$	$r_3 = 0$

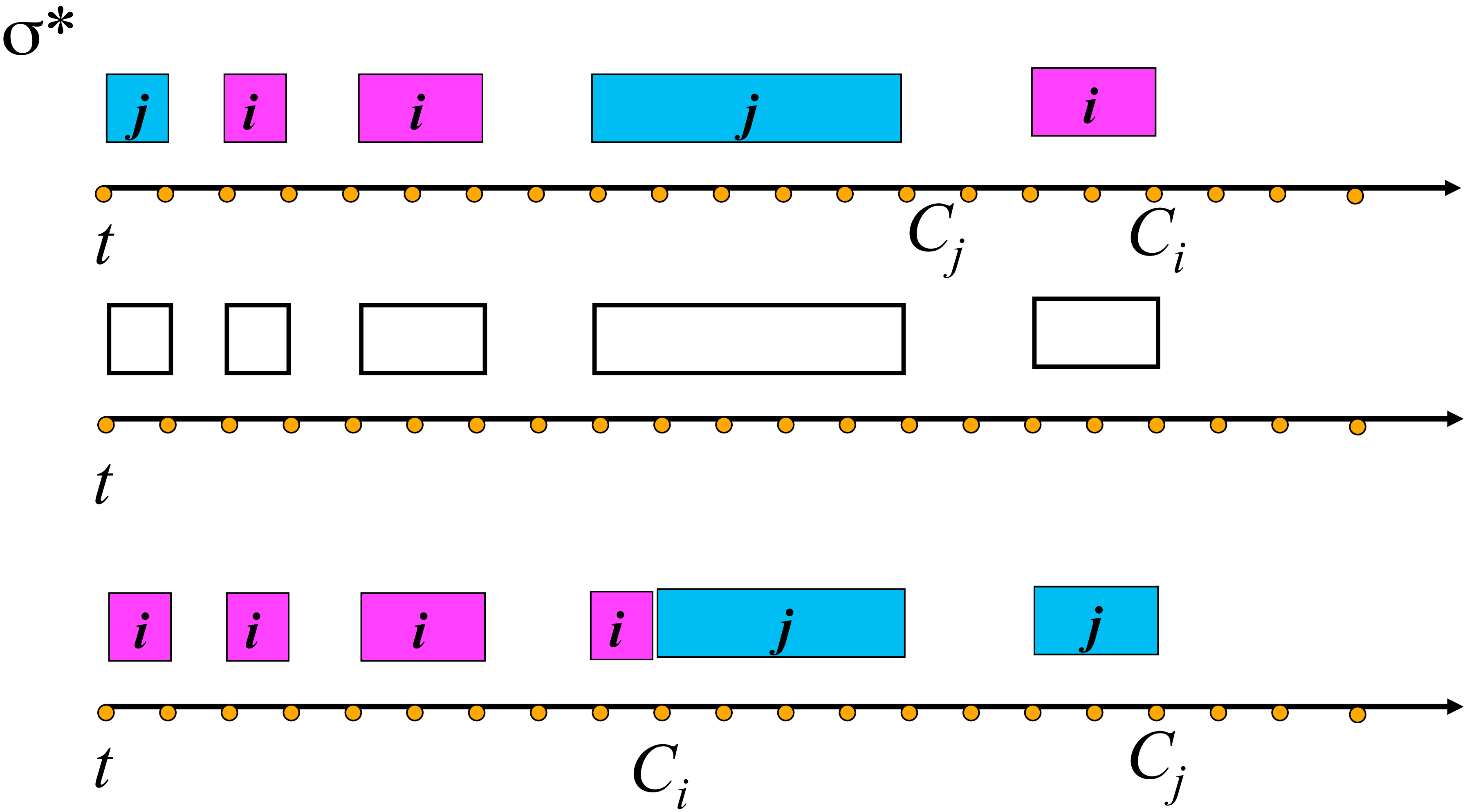
$1|pmtn, r_j|\Sigma C_j$

- We will show that we can convert any ***preemptive*** schedule into a nonpreemptive schedule in such way that the completion time of each job at most doubles.
- In a preemptive schedule, we can still 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 job.

SRPT rule

- Each time that a job is completed, or at the next release date, the job to be processed next has the smallest remaining processing time among the available jobs.
- Denote by σ the schedule obtained by SRPT rule and show that σ is optimal.
- Assume that an optimal schedule σ^* coincides with a schedule σ up to time t.

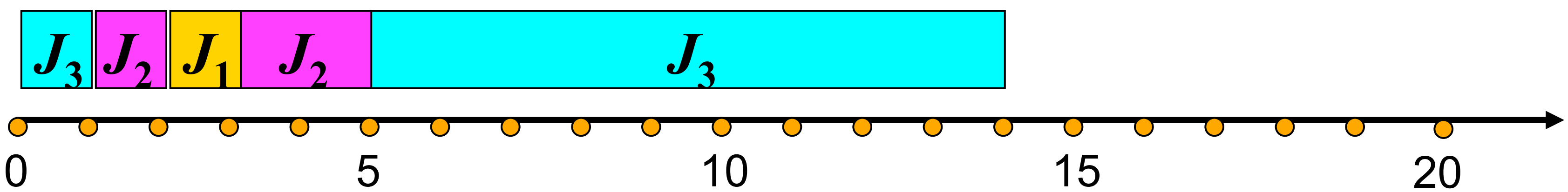
Exchange argument



Preemptive solution

$$C_1(\sigma^{pr}) + C_2(\sigma^{pr}) + C_3(\sigma^{pr}) = 22$$

σ^{pr}



J_1	$p_1 = 1$	$r_1 = 2$
J_2	$p_2 = 3$	$r_2 = 1$
J_3	$p_3 = 10$	$r_3 = 0$

Lower bound

- Let $C_j(\sigma^{\mathrm{pr}})$ be the completion time of job j in an optimal preemptive schedule.
- Let OPT be the sum of completion times in an optimal nonpreemptive schedule.
- We have

$$\sum_{j=1}^{n} C_j\left(\sigma^{\mathrm{pr}}\right) \leq \mathrm{OPT}.$$

Algorithm «Rounding preemptive schedule»

1. Find an optimal preemptive schedule σ^{pr} using SRPT.
2. Schedule the jobs in σ nonpreemptively in the same order that they complete in σ^{pr}.

To be more precise, suppose that the jobs are indexed such that $C_1(\sigma^{pr}) \le C_2(\sigma^{pr}) \le \ldots \le C_n(\sigma^{pr})$. 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.

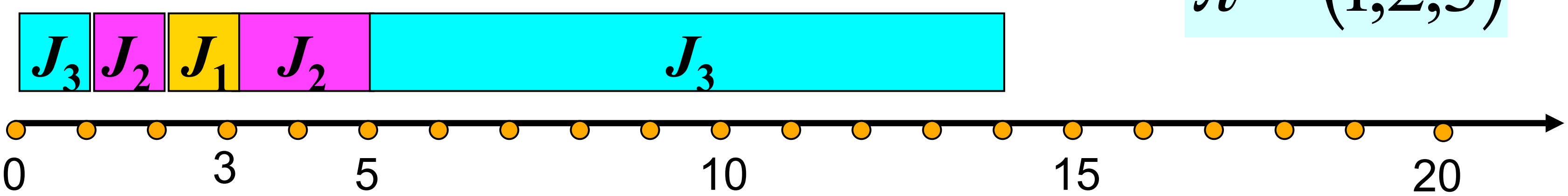
We will show that scheduling nonpreemptively in this way does not delay the jobs by too much.

Example

$$C_1(\sigma^{pr}) + C_2(\sigma^{pr}) + C_3(\sigma^{pr}) = 22$$

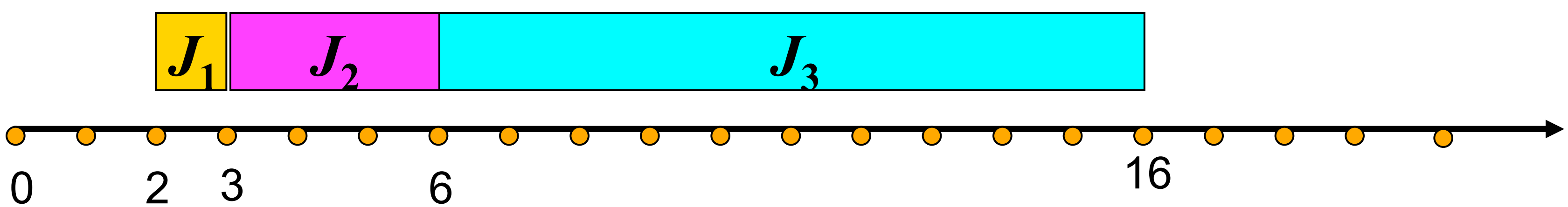
σ^{pr}

$$\pi = (1,2,3)$$



$$C_1(\sigma) + C_2(\sigma) + C_3(\sigma) = 25$$

σ



Lemma 11.1

For each job $j = 1,\ldots,n$, $C_j(\sigma) \leq 2C_j(\sigma^{pr})$.

Proof

- Let us first derive some easy lower bounds on $C_j(\sigma^{\mathrm{pr}})$. Since we know that j is processed in σ^{pr} after jobs 1,…, j−1, we have

$$C_j\left(\sigma^{\mathrm{pr}}\right) \geq \max_{k=1,\dots,j} r_k, \quad C_j\left(\sigma^{\mathrm{pr}}\right) \geq \sum_{k=1}^{j} p_k.$$

- By construction it is also the case that

$$C_j\left(\sigma\right) \geq \max_{k=1,\dots,j} r_k.$$

Proof

- 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 $\left[\max_{k=1,\ldots,j} r_k, C_j(\sigma)\right]$, there cannot be any point in time at which the machine is idle.
- Therefore, this interval can be of length at most $\sum_{k=1}^{j} p_k$.

$$C_j(\sigma) \leq \max_{k=1,\ldots,j} r_k + \sum_{k=1}^{j} p_k \leq 2C_j(\sigma^{\mathrm{pr}}).$$

2-approximation

Theorem 11.2

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

$$\sum_{j=1}^{n} C_j(\sigma) \le 2\sum_{j=1}^{n} C_j\left(\sigma^{\mathrm{pr}}\right) \le 2\,\mathrm{OPT}.$$

$1|r_j|\Sigma w_j C_j$

- Single machine
- $\boldsymbol{J} = \{1,\ldots, n\}$ – jobs
- $p_j \geq 0$ – processing time of job j.
- $r_j \geq 0$ – release time of job j.
- $w_j \geq 0$ – weight of job j.
- $C_j(\sigma)$ – completion time of job j in σ.
- No preemption.
- The machine cannot process two jobs at the same time.

$1|\text{pmtn}, r_j|\Sigma w_j C_j$

- The algorithm "Rounding preemptive schedule" and analysis give us a way to round any preemptive schedule to one whose sum of weighted completion times is at most twice more.
- Unfortunately, we cannot use the same technique of finding a lower bound on the cost of the optimal nonpreemptive schedule by finding an optimal preemptive schedule.
- Unlike the unweighted case, it is NP-hard to find an optimal schedule for the preemptive version of the weighted case.

What we use to obtain the 2-approximation?

$$C_j\left(\sigma^{\mathrm{pr}}\right) \geq \max_{k=1,\ldots,j} r_k$$

$$C_j\left(\sigma^{\mathrm{pr}}\right) \geq \sum_{k=1}^{j} p_k$$

$$\sum_{j=1}^{n} C_j\left(\sigma^{\mathrm{pr}}\right) \leq \mathrm{OPT}$$

What we use to obtain the 2-approximation?

$$C_j\left(\sigma^{\mathrm{pr}}\right) \geq \max_{k=1,\ldots,j} r_k \qquad C_j\left(\sigma^{\mathrm{pr}}\right) \geq \alpha \max_{k=1,\ldots,j} r_k$$

$$C_j\left(\sigma^{\mathrm{pr}}\right) \geq \sum_{k=1}^{j} p_k \qquad C_j\left(\sigma^{\mathrm{pr}}\right) \geq \beta \sum_{k=1}^{j} p_k$$

$$\sum_{j=1}^{n} C_j\left(\sigma^{\mathrm{pr}}\right) \leq \mathrm{OPT} \qquad \sum_{j=1}^{n} C_j\left(\sigma^{*}\right) \leq \mathrm{OPT}$$

We can give a linear programming 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 for the $1|r_j|\Sigma w_j C_j$ problem.

Variables and constraints

- Denote by C_j the completion time of job j.
- We want to minimize $\sum_{j \in S} w_j C_j$.

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

Second set of constraints

- Consider some set $S \subseteq \boldsymbol{J}$ of jobs and the sum $\sum_{j \in S} p_j C_j$.
- This sum is minimized when all the jobs in S have a release date of 0 and all the jobs in S finish first in the schedule.
- Assuming these two conditions hold, then any completion time $C_j(\sigma)$ for $j \in S$ is equal to p_j + the sum of all 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 of all jobs in S that preceded j in the schedule.
- The sum $\sum_{j \in S} p_j C_j$ must contain $p_j p_k$ for all pairs $j,k \in S$.

Queyranne's inequality

$$\sum_{j \in S} p_j C_j = \sum_{j,k \in S : j \leq k} p_j p_k =$$

$$= \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2 + \frac{1}{2}\sum_{j \in S} p_j^2 \geq \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2$$

$\mathrm{LP}(1|r_j|\Sigma w_j C_j)$

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

$$\text{s.t.} \qquad C_j \geq r_j + p_j, \qquad \forall j \in J, \qquad (1)$$

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2, \ \forall S \subseteq J. \qquad (2)$$

Algorithm $1|r_j|\Sigma w_j C_j$

1. Find an optimal solution $\sigma^* = (C_1(\sigma^*), C_2(\sigma^*), \ldots, C_n(\sigma^*))$ of the LP($1|r_j|\Sigma w_j C_j$).
2. Schedule the jobs in σ nonpreemptively in the same order that they complete in σ*.
3. **Output** (σ)

3-approximation

Theorem 11.3

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

Proof

$$\sum_{j=1}^{n} w_j C_j^* \le \mathrm{OPT}.$$

- Assume that the jobs are reindexed so that $C_1(\sigma^*) \le C_2(\sigma^*) \le \ldots \le C_n(\sigma^*)$.
- As in the proof of Lemma 11.1, there cannot be any idle time in the time interval $\left[\max_{k=1,\ldots,j} r_k, C_j(\sigma)\right]$.
- Therefore it must be the case that

$$C_j(\sigma) \le \max_{k=1,\ldots,j} r_k + \sum_{k=1}^{j} p_k.$$

$$C_j(\sigma) \leq \max_{k=1,\dots,j} r_k + \sum_{k=1}^{j} p_k$$

- Let $l \in \{1,\dots,j\}$ be the index of the job that maximizes $\max_{k=1,\dots,j} r_k$ so that $r_l = \max_{k=1,\dots,j} r_k$.
- We have $C_j(\sigma^*) \geq C_l(\sigma^*)$ and $C_l(\sigma^*) \geq r_l$ by the LP constraints; thus $C_j(\sigma^*) \geq \max_{k=1,\dots,j} r_k$.
- Consider set $S = \{1,\dots,j\}$.
- From the fact that σ^* is a feasible LP solution , we know that
$$\sum_{k \in S} p_k C_k(\sigma^*) \geq \frac{1}{2}\left(\sum_{k \in S} p_k\right)^2.$$
- Since $C_1(\sigma^*) \leq C_2(\sigma^*) \leq \dots \leq C_j(\sigma^*)$, we have
$$C_j(\sigma^*) \sum_{k \in S} p_k \geq \sum_{k \in S} p_k C_k(\sigma^*) \geq \frac{1}{2}\left(\sum_{k \in S} p_k\right)^2.$$
- By combining these two inequalities we see that $C_j(\sigma^*) \geq \frac{1}{2} \sum_{k \in S} p_k$.

Proof

$$C_j(\sigma^*) \geq \max_{k=1,\ldots,j} r_k \qquad C_j(\sigma^*) \geq \frac{1}{2}\sum_{k\in S} p_k.$$

$$C_j(\sigma) \leq \max_{k=1,\ldots,j} r_k + \sum_{k=1}^{j} p_k \leq C_j(\sigma^*) + 2C_j(\sigma^*) = 3C_j(\sigma^*).$$

$$\sum_{j=1}^{n} w_j C_j(\sigma) \leq 3\sum_{j=1}^{n} w_j C_j(\sigma^*) \leq 3\,\mathrm{OPT}.$$

How to solve LP?

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

$$\text{s.t.} \quad C_j \geq r_j + p_j, \qquad \forall j \in J \qquad (1),$$

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2, \ \forall S \subseteq J \qquad (2).$$

Ellipsoid method (draft)

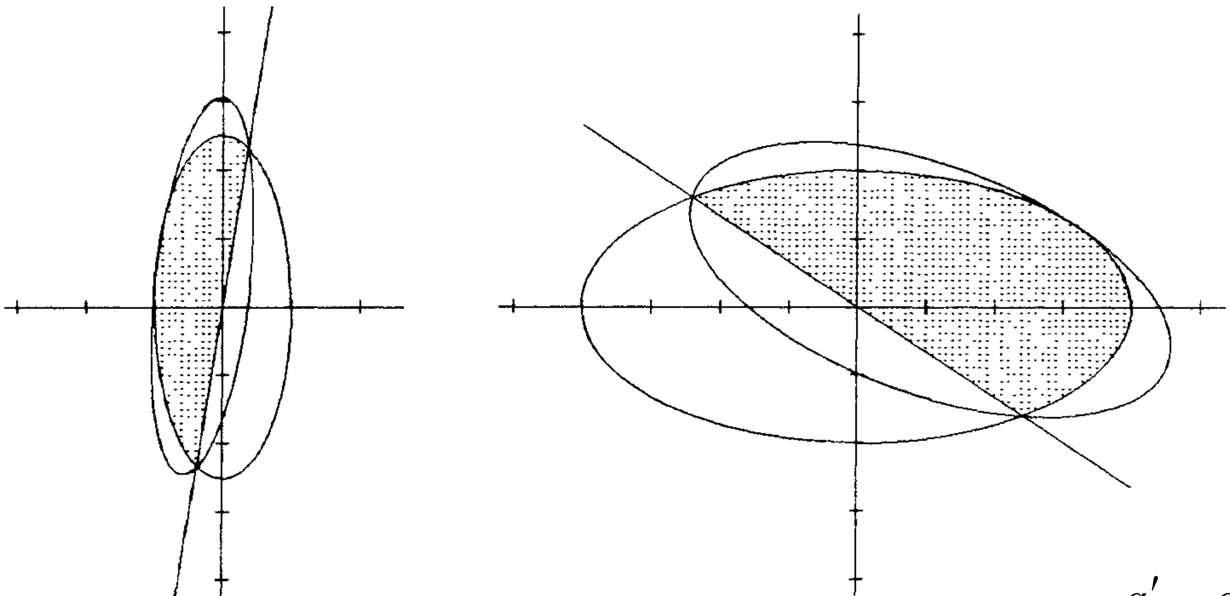
- The input for the algorithm is a system of inequalities $P = \{Cx \le d\}$ with n variables in integral coefficients.
- We would like to determine whether P is empty or not, and if it is nonempty, we would like to find a point in P.

1. Let $N{=}2n((2n{+}1)\langle C\rangle + n\langle d\rangle - n^3)$ and $k = 0$
2. Find a "big" ellipsoid $E_0(A_0,a_0)$, that contains our polytope P.
3. **If** $k = N$, **then** STOP! (Declare P empty.)
4. **If** $a_k \in P$, **then** STOP! (A feasible solution is found.)
5. **If** $a_k \notin P$, **then** choose an inequality that is violated by a_k.
6. Create a new ellipsoid $E_{k+1}(A_{k+1},a_{k+1})$, go to 3.

Löwner-John ellipsoid

$$E = E(A,a) = \{x \in \boldsymbol{R}^n \mid (x-a)^T A^{-1}(x-a) \leq 1\}$$

$$E'(A,a,c) = E(A,a) \cap \{x \in \boldsymbol{R}^n \mid c^T x \leq c^T a\}$$



(a) (b)

Figure 3.3

$$a' = a - \frac{1}{n+1} b, \quad \left(b = \frac{1}{\sqrt{c^T A c}} A c \right)$$

$$A' = \frac{n^2}{n^2 - 1} \left(A - \frac{2}{n+1} b b^T \right)$$

Ellipsoid method (draft)

- The input for the algorithm is a system of inequalities $P=\{Cx \le d\}$ with n variables in integral coefficients.
- We would like to determine whether P is empty or not, and if it is nonempty, we would like to find a point in P.

1. Let $N=2n((2n+1)\langle C\rangle + n\langle d\rangle - n^3)$ and $k = 0$
2. Find a "big" ellipsoid $E_0(A_0,a_0)$, that contains our polytope P.
3. **If** $k = N$, **then** STOP! (Declare P empty.)
4. **If** $a_k \in P$, **then** STOP! (A feasible solution is found.)
5. **If** $a_k \notin P$, **then** choose an inequality that is violated by a_k.
6. Create a new ellipsoid $E_{k+1}(A_{k+1},a_{k+1})$, go to 3.

We need a polynomial time procedure (separation oracle) for steps 4 and 5.

How to find the violated constraint?

$$\sum_{j\in S} p_j C_j \geq \frac{1}{2}\left(\sum_{j\in S} p_j\right)^2, \ \forall S \subseteq J\,?$$

- Given a solution σ.
- Reindex the variables so that $C_1(\sigma) \leq C_2(\sigma) \leq \ldots \leq C_n(\sigma)$.
- Let $S_1=\{1\}$, $S_2=\{1, 2\}$,…, $S_n=\{1,\ldots, n\}$.
- We claim that it is sufficient to check whether the constraints are violated for the n sets S_1, S_2,…, 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.

Separation oracle

Lemma 11.4

Given variables C_j, if constraints (2) are satisfied for the n sets S_1, S_2,…, S_n, thet they are satisfied for all $S \subseteq \boldsymbol{J}$.

Proof (1)

- Let $S \subseteq \boldsymbol{J}$ be a constraint that is not satisfied; that is
$$\sum_{j \in S} p_j C_j < \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2.$$
- 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
$$x = \sum_{j \in S} p_j C_j - \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2$$
- Any such change will result in another set S' that also does not satisfy the constraint.

$$x = \sum_{j \in S} p_j C_j - \frac{1}{2}\left(\sum_{j \in S} p_j\right)^2$$

- Removing a job k from S decreases x if

$$-p_k C_k + p_k \sum_{j \in S \setminus \{k\}} p_j + \tfrac{1}{2} p_k^2 < 0 \Leftrightarrow C_k > \sum_{j \in S \setminus \{k\}} p_j + \tfrac{1}{2} p_k.$$

- Adding a job k to S decreases x if

$$p_k C_k - p_k \sum_{j \in S} p_j - \tfrac{1}{2} p_k^2 < 0 \Leftrightarrow C_k < \sum_{j \in S} p_j + \tfrac{1}{2} p_k.$$

Removing of jobs

- Let l be the highest indexed job in S.
- We remove l from S if $C_l > \sum_{j \in S \setminus \{l\}} p_j + \frac{1}{2} p_l;$
- In this case the resulting set $S \setminus \{l\}$ also does not satisfy the constraint (2).
- 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 l has $C_l \leq \sum_{j \in S \setminus \{l\}} p_j + \frac{1}{2} p_l.$

Adding of jobs

- Now suppose $S' \neq S_l = \{1, \ldots, l\}$.
- Let $k < l$ and $k \notin S_l$.
- We have $C_k \leq C_l \leq \sum_{j \in S \setminus \{l\}} p_j + \frac{1}{2} p_l < \sum_{j \in S} p_j < \sum_{j \in S} p_j + \frac{1}{2} p_k.$
- It follows that, adding k to S' can only decrease the difference $$x = \sum_{j \in S} p_j C_j - \frac{1}{2} \left(\sum_{j \in S} p_j \right)^2 .$$
- Thus we can add all $k < l$ to S', and the resulting set S_l will also not satisfy the constraint (2).

Ellipsoid Method (1)

- Suppose we are trying to solve LP($1|r_j|\Sigma w_j C_j$).
- Initially, the algorithm finds an ellipsoid in $\mathbf{R}^n$ containing all basic solutions for the linear program.
- Let $\check{C}$ be the center of the ellipsoid.
- The algorithm calls the separation oracle with $\check{C}$.
- If $\check{C}$ is feasible, it creates a constraint $\Sigma w_j C_j \leq \Sigma w_j \check{C}_j$, since a basic optimal solution must have objective function value no greater than the feasible solution $\check{C}$.
- This constraint is sometimes called **an objective function cut**.

Ellipsoid Method (2)

- If $\check{C}$ is not feasible the separation oracle returns a constraint $\Sigma a_{ij}C_j \geq b_i$ that is violated by $\check{C}$.
- In either case, we have a hyperplane through $\check{C}$ such that a basic optimal solution to the linear program must lie on one side of the hyperplane.
- In the case of a feasible $\check{C}$ the hyperplane is $\Sigma w_j C_j \leq \Sigma w_j \check{C}_j$.
- In the case of an infeasible the $\check{C}$ the hyperplane is $\Sigma a_{ij}C_j \geq \Sigma a_{ij}\check{C}_j$.

Ellipsoid Method (3)

- The hyperplane containing $\check{C}$ splits the ellipsoid in two.
- The algorithm then finds a new ellipsoid containing the appropriate half of the original ellipsoid, and then consider the center of new ellipsoid.

Löwner-John ellipsoid

$$E = E(A,a) = \{x \in \boldsymbol{R}^n \mid (x-a)^T A^{-1}(x-a) \le 1\}$$

$$E'(A,a,c) = E(A,a) \cap \{x \in \boldsymbol{R}^n \mid c^T x \le c^T a\}$$

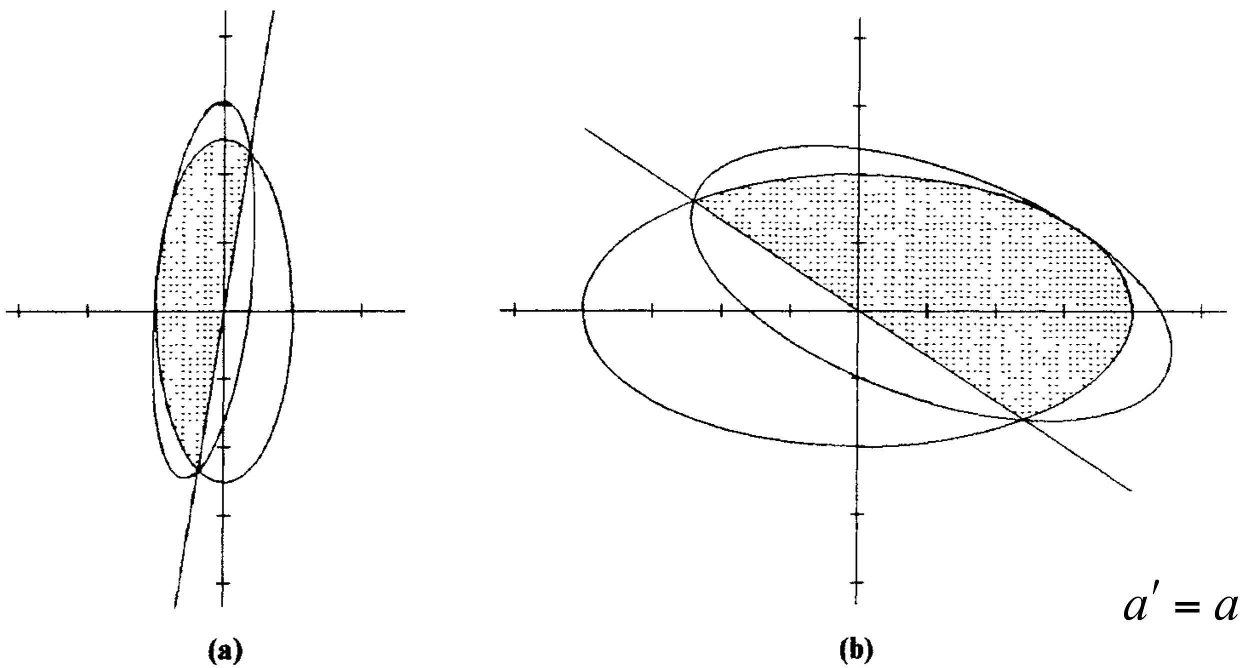


Figure 3.3

$$a' = a - \frac{1}{n+1} b, \quad \left(b = \frac{1}{\sqrt{c^T A c}} A c \right)$$

$$A' = \frac{n^2}{n^2 - 1} \left(A - \frac{2}{n+1} b b^T \right)$$

Ellipsoid Method (3)

- The hyperplane containing containing $\check{C}$ splits the ellipsoid in two.
- The algorithm then finds a new ellipsoid containing the appropriate half of the original ellipsoid, and then consider the center of new ellipsoid.
- This process repeats until the ellipsoid is sufficiently small that it can contain at most one basic feasible solution.
- This solution must be a basic optimal solution.

Exercise

- Consider a single machine scheduling problem $1|\text{prec}|\Sigma w_j C_j$ in which we have precedence constraints but no release dates. We say i **precedes** j if in any feasible schedule, job i must be completely processed before job j begins processing.
- We are given n jobs with processing times $p_j > 0$ and weights $w_j > 0$, and the goal to find a nonpreemptive schedule on a single machine that is feasible respect to the precedence constraints and that minimizes the weighted sum of completion times of jobs.
- Design LP relaxation of $1|\text{prec}|\Sigma w_j C_j$ and give a 2-approximation algorithm for this problem.