(4) Approximation algorithms

Notation:

We will refer to scheduling problems using triplet notation:

$\alpha \mid \beta \mid \gamma$

- α: machine environment
- β: Side Constraints
- γ: Objective (will usually be a min. objective, so we will omit 'min :)

Ex:

α (M/C environment)
- 1: Denotes single M/C
- P: Multiple identical M/C's
- P2 or Pm: m identical M/C's

(... and more to come)

γ (Objective Function)
- $\sum C_j$, $\sum w_j C_j$, L_{max}, f_{max} (Things we have seen)
- $\sum f_j(C_j)$, $C_{max} := \max_j C_j$ (Other possible objective functions)

Note: This is trivial with a single machine

β (Side constraints)
- This can be empty (which is exactly what we've seen so far in the BSA setup)
  i.e. $1 \mid\mid \sum C_j$, $1 \mid\mid \sum w_j C_j$, $1 \mid\mid L_{max}$, $1 \mid\mid f_{max}$.
- There are other constraints we will see:
  - r_j: Release Dates
    Specifies that job j becomes available at time r_j
    (i.e. Cannot schedule j at time $< r_j$)

Continued

Ex: (continued)

- Prec: Precedence Constraint

A precedence constraint $j \to k$ specifies that j must complete before k can be started

Note:

Precedence is a transitive relation i.e. $j \to k$, $k \to l \Rightarrow j \to l$.

This also implies that we cannot have a cycle of precedence constraints:

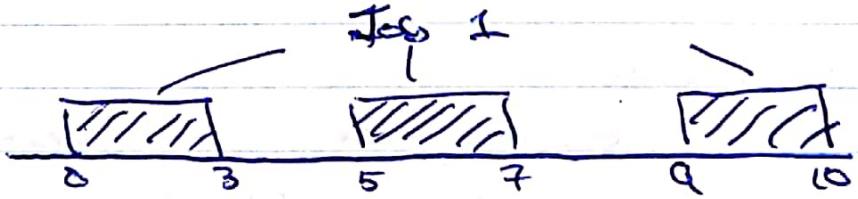
i.e. $j_1 \to j_2 \to \ldots \to j_k \to j_1$ is not possible)

Precedence constraints are represented by DAGs (i.e. Nodes := jobs, Arcs := Constraints)

- Pmtn/Pmp: Preemption

Can interrupt a job and resuming working on it later.

i.e. We can have:



as a valid schedule.

$1|prec|f_{max}$:

By our notation, this is the problem:

1 MC, min. f_{max}, and subject to precedence constraints $j \to k$.

We define:

Given a set J of jobs, let

$$L(J) := \{j \in J : j \text{ has no successors in } J\}$$
$$:= \{k \in J \text{ s.t. } j \to k\}$$

Observe: Only jobs in $L(J)$ can appear at the end of a schedule for J.

Def'n 4.1 (Modified LCL for $1|prec|f_{max}$)

- $J \leftarrow \{1, \ldots, n\}$
- while $J \neq \emptyset$:
  - find $\ell \in L(J)$ s.t. $f_\ell(\sum_{j \in J} p_j) = \min_{k \in L(J)} f_k(\sum_{j \in J} p_j)$
  - Schedule ℓ last
  - $J \leftarrow J \setminus \{\ell\}$ (and update $L(J)$).

Theorem 4.1:

The above algorithm gives an optimal schedule for $1|prec|f_{max}$.

[Proof]

Similar to the proof of Thm 2.4. □

↳ $1||r_j, prmtn|\sum C_j$

$1|r_j, prmtn|\sum C_j$

We have:

1 M/C, min $\sum C_j$, and with the constraints of release dates and allowing preemption.

(Note: without r_j, preemption is unnecessary)

Defn 4.2 (Shortest Remaining Processing Time Rule - SRPT)

At each point of time, schedule the job available with smallest remaining processing time, preempting a job if a job with smaller processing time is released

Theorem 4.2:

SRPT generates an optimal schedule for $1|r_j, prmtn|\sum C_j$.

[Proof]

Let S^* be an optimal schedule that is NOT an SRPT schedule.

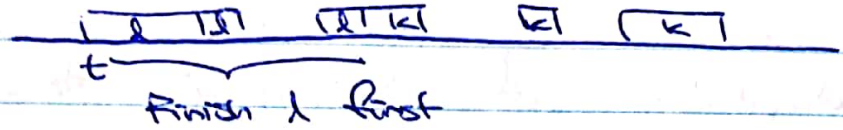
Then, at some point of time t, S^* schedules some job k with remaining proc time x_k, but there was another available job l with remaining proc time $x_l < x_k$ and $x_l > 0$

Continued

[Proof] (continued)

S^*: (Placement of k, l arbitrary)

S:



We want to be careful with our interchange argument (Else we might affect the completion time of other jobs)

Consider portions of time after t where k or l are scheduled in S^*, and interchange k, l by scheduling l in the first x_l of the time, followed by k to get schedule S.

This ensures that $C_j^S = C_j^{S^*}$ $\forall j \neq k, l$, and also $C_k^S = \max(C_k^{S^*}, C_l^{S^*})$.

Claim 1: $C_l^S < C_k^{S^*}$

Since k is processed for time $x_k > x_l$.

Claim 2: $C_l^S < C_l^{S^*}$

l is not scheduled at time t in S^*, so $C_l^{S^*}$ will be pushed back.

So, $C_l^S < \min(C_k^{S^*}, C_l^{S^*})$

$\Rightarrow C_l^S \cdot C_k^S < C_l^{S^*} \cdot C_k^{S^*}$

Completing the contradiction

□

$1|r_j|\sum C_j$:

Now, we look at the previous problem, but without preemption

Notice that SRPT generalizes SPT, so:

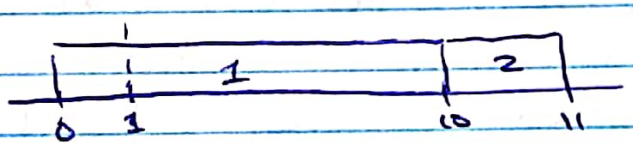
Q: Can we come up with generalizations or adaptations of SRPT / SPT to solve $1|r_j|\sum C_j$?

Attempt 1!

At each time t, schedule the available job with shortest processing time, but do not preempt

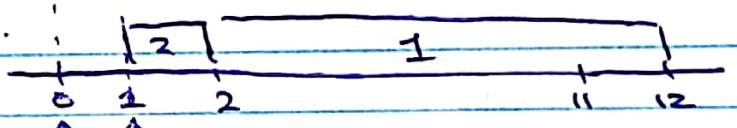
However, consider:

S:



$\sum C_j^S = 10+11 = 21$

S^*:



$\sum C_j^{S^*} = 2+12 = 14$.

Job 2: $r_j = 1$, $p_j = 1$

Job 1: $r_j = 0$, $p_j = 10$

Notice that S^* is more optimal than S, so our algorithm is no longer optimal.

The problem is: A small job got released soon after a big job was started

→ Attempt 2

Attempt 2!

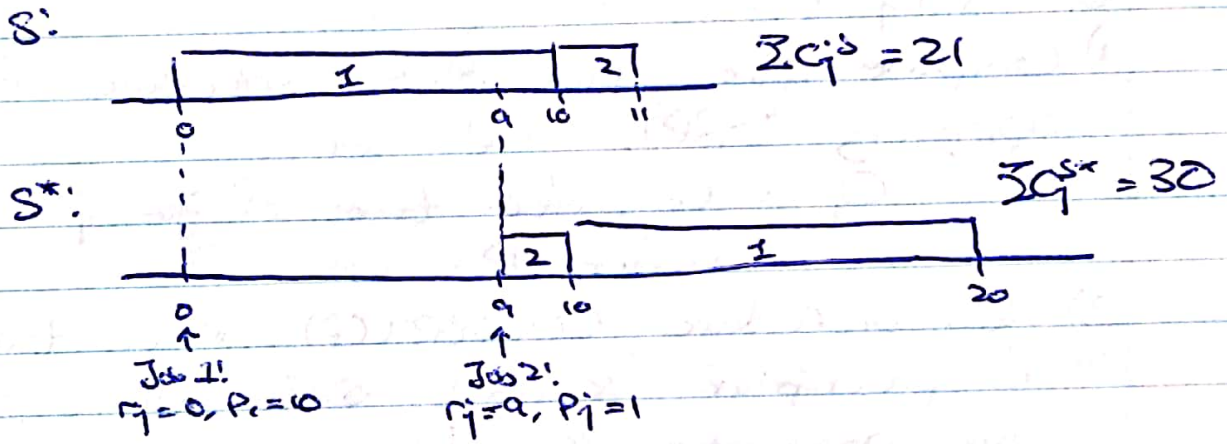
We might try to fix our previous example.

Consider:

Sort jobs in increasing p_j order, and process them in this order, waiting for job j, if j is not released by the completion time of the previous job.

(Notice that this suggests that it may be beneficial to leave the machine idle even if there are available jobs - called "enforced idle time")

However, Consider:



Again, our algorithm doesn't work since S is more optimal than S^*.

(An optimal solution to $1|r_j|\sum C_j$ is not known!)

What can we do?

- Abandon Efficiency
- Abandon Optimality.

Continued

Defn 4.3 (α-approximation algorithm)
An α-approximation algorithm is an efficient algorithm that returns a schedule S whose objective value ≤ α · (optimum) (where optimum is the optimal objective value for our given problem, and we will commonly write it as OPT)
Further, α is known as the "approximation ratio" or "approximation guarantee".

We will design a 2-approximation algorithm for $1|r_j|\sum C_j$:
1) Compute the preemptive schedule P by applying SRPT, and let
C_j^P := Completion time of job j under P.
2) Run procedure CONVERT(P), which takes a preemptive schedule and returns a non-preemptive schedule

CONVERT(P):
1) Sort jobs in increasing order of C_j^P:
$C_1^P \le C_2^P \le \dots \le C_n^P$
2) Schedule jobs "non-preemptively" in the order defined above, and output the resulting schedule
(By "non-preemptively", we mean:
Start Job j at time
$t := \max(\text{Completion time of } j-1, r_j)$).

Theorem 4.3:
SRPT + CONVERT is a 2-approx. algorithm for $1|r_j|\sum C_j$.

To prove this, we need to following:

Claim 4.4:
Let OPT := optimal value for $1|r_j|\sum C_j$.
Then:

$$OPT \geq \sum C_j^P$$

Proof]
We've proved that SRPT gives an optimal schedule for $1|r_j, pmtn|\sum C_j$ (Theorem 4.2), so:

$$\sum C_j^P = OPT_{1|r_j, pmtn|\sum C_j}$$

And:

$$OPT_{1|r_j, pmtn|\sum C_j} \leq OPT$$

Since any schedule for $1|r_j|\sum C_j$ is feasible for $1|r_j, pmtn|\sum C_j$. The preemptive problem is a relaxation of $1|r_j|\sum C_j$. □

Lemma 4.5:
For every job j (in the ordering after step (1) of CONVERT):

$$C_j^S \leq 2C_j^P$$

← Proof

[Proof]

We begin with the following claim:

For every jobs j (under the order after Step 1):

$$C_j^P \geq \max_{k=1,\ldots,j} r_k \quad , \text{and} \quad (1)$$

$$C_j^P \geq \sum_{k=1}^{j} p_k \quad (2)$$

Since by definition:

$$C_j^P \geq C_k^P \quad \forall\, k=1,\ldots,j$$

$$\geq r_k \quad \forall\, k=1,\ldots,j$$

$$\geq \max_{k=1,\ldots,j} r_k$$

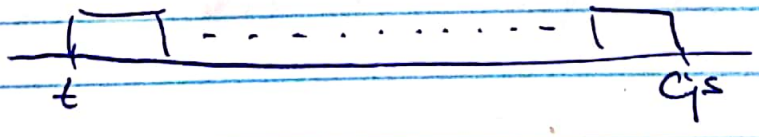
And, by time C_j^P, all jobs $1,\ldots,j$ have completed in the preemptive schedule P

So,

$$C_j^P \geq \sum_{k=1}^{j} p_k$$

Now, in schedule S, let $t = \max_{k=1,\ldots,j} r_k$, and [redacted], consider the interval $[t, C_j^S]$

S:



Goal: We want to bound this interval by C_j^P

→ Continued

[Proof] (continued)
Notice that:
- There is no idle time in $[t, C_j^S]$. Since the only way this can happen is if we are waiting for a job to be released. However, by time t, all jobs $1, \dots, j$ have been released and further, we would complete j before waiting for any other job l with $l > j$.
- This also means that in $[t, C_j^S]$, we are only processing jobs $1, \dots, j$.

Hence,

$$C_j^S = t + (\text{length of } [t, C_j^S])$$

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

(By the 2nd point)

$$\leq \underset{(1)}{C_j^P} + \underset{(2)}{C_j^P} = 2C_j^P \qquad \square$$

Note:
If we can solve a preemptive version of a non-preemptive problem, CONVERT and this lemma gives a strong guarantee for the completion times.

Now, we can prove Theorem 4.3:
[Proof] (of Theorem 4.3):
By Lemma 4.5, we get that:

$$\sum_j C_j^S \leq 2 \sum_j C_j^P$$

And by claim 4.4:

$$\sum_j C_j^S \leq 2 \sum_j C_j^P \leq 2 \cdot OPT.$$

$\square$.

Hilroy

Runtime Analysis:

SRPT requires:

- $O(n^2)$:
  - The schedule changes at most $2n$ time points corresponding to completion times + release dates of jobs
  - At each of these time points, we go through the list of remaining jobs in $O(n)$ time

  So, $O(n \cdot 2n) = O(n^2)$
- $O(n \log n)$
  - By using a min-heap or a sorted list, we can reduce the $O(n)$ lookup to $O(\log n)$

  So, $O(n \cdot \log n)$

Then, our 2-approx. alg takes:

SRPT + CONVERT

$O(n \log n)$ $O(n \log n)$ (sorting)

$= O(n \log n)$ time.

$1|r_j|\sum w_jC_j$ and $1|r_j, pmtn|\sum w_jC_j$

We know that $1|r_j, pmtn|\sum w_jC_j$ is a relaxation of $1|r_j|\sum w_jC_j$ and we have a general strategy to tackle preemptive problems and convert them to their non-preemptive counterparts. So, we want an α-approx. algorithm A for $1|r_j, pmtn|\sum w_jC_j$ so that A + CONVERT gives an 2α-approx algorithm for $1|r_j|\sum w_jC_j$.

Preemptive WSPT:
At each point of time, schedule available jobs with largest w_j/p_j ratio, preempting as necessary.
Equivalently:
1) Order jobs in decreasing order of density
2) Schedule jobs, preemptively, in this order
(i.e. At each point of time, schedule jobs that came earliest in this order)

Note: Preemptive WSPT does not coincide with SPT, even when all jobs have weight 1, since p_j is the original processing time for j.

Theorem 4.6:
Preemptive WSPT is a 2-approx. algorithm for $1|r_j, pmtn|\sum w_jC_j$.

→ proof.

Hilroy

[Proof]

Let $OPT := OPT_{1|r_j, pmtn|\sum w_j C_j}$

Let jobs be ordered $1, \ldots, n$ so that:

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

and S be the schedule returned by preemptive WSPT

Observe that $OPT \geq \sum_{j=1}^{n} w_j C_j^{(1)}$. Since the completion time $C_j \geq r_j$ in any schedule and, $OPT \geq OPT_{1|pmtn|\sum w_j C_j}$, which is the relaxation of the problem, with all release dates of 0. (Any schedule for $1|r_j, pmtn|\sum w_j C_j$ is a schedule $1|pmtn|\sum w_j C_j$)

Further $1|pmtn|\sum w_j C_j \equiv 1||\sum w_j C_j$, since preemption is unnecessary when $r_j = 0 \; \forall j$. So we can use WSPT, which produces an optimal schedule with objective value $OPT_{1||\sum w_j C_j}$. The schedule will also be exactly in the order $1, \ldots, n$, so:

$$\begin{aligned} OPT &\geq OPT_{1|pmtn|\sum w_j C_j} \\ &= OPT_{1||\sum w_j C_j} \\ &= \sum_{j=1}^{n} w_j \left(\sum_{k=1}^{j} p_k\right). \qquad (2) \end{aligned}$$

C6 continued

[Proof] (Continued)
We are nearly done, Consider schedule S and job J:

S:

In the interval $[r_j, C_j^S]$, there is never any idle time, since job j is available. And, only jobs k $(1 \le k \le j)$ can be scheduled, since any job l with $l \ge j+1$ would contradict our ordering.
So:

$$C_j^S \le r_j + \sum_{k=1}^{j} p_k \quad \forall j = 1, \ldots, n$$

Then:

$$\sum_{j=1}^{n} w_j C_j^S \le \sum_{j=1}^{n} w_j \left(r_j + \sum_{k=1}^{j} p_k\right) \quad \text{(By above)}$$

$$= \sum_{j=1}^{n} w_j r_j + \sum_{j=1}^{n} w_j \left(\sum_{k=1}^{j} p_k\right)$$

$$\le \underset{(1)}{OPT} + \underset{(2)}{OPT} = 2 \cdot OPT$$

□.

And so, Preemptive WSPT + CONVERT gives an algorithm for $1|r_j|\sum w_j C_j$.

Corollary 4.7:
Preemptive WSPT + CONVERT is a 4-approx. alg for $1|r_j|\sum w_j C_j$