ADM III WiSe 2014/15
Scheduling and Project Planning

LaTeX by Martin Plonka

November 1, 2015

Contents

1 Projects and Partial Orders

Definition 1.1:
*A **project** consists of for example*

(1) activities / jobs u, v, i, j or just $V = \{1, \ldots, n\}$

(2) job data

- *processing time / duration (random or deterministic)*
- *resource requirements*
- *processing costs*

(3) project data

- *available resources*
- *limited budget*
- *deadlines*

(4) project rules

- *temporal conditions*
- *resource conditions*

The simplest case consists only of precedence conditions. Therefore we define a partial order.

Definition 1.2:
We define the binary relation $<$ on V by

$$i \text{ precedes } j \qquad i < j \qquad :\Leftrightarrow \qquad j \text{ must wait for } i$$

*meaning i precedes j. This defines a **partial order**:*

$$i < j \Rightarrow j \not< i \qquad \text{(asymmetric)}$$
$$i < j \wedge j < k \Rightarrow i < k \qquad \text{(transitive)}$$

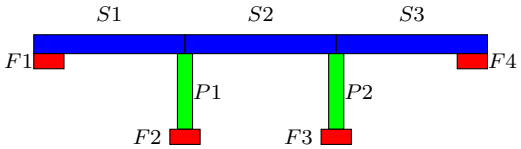
We may represent our projects as activity on node diagram, meaning an directed acyclic graph $G = (V, E)$. Where E is the transitive reduction of precedence constraints $<$.

Definition 1.3:

- $Pred_G(i) := \{j \in V \mid j < i\}$
- $Suc_G(i) := \{j \in V \mid i < j\}$
- $ImPred_G(i) := \{j \in V \mid j < i \wedge (j, i) \in E\}$
- $ImSuc_G(i) := \{j \in V \mid i < j \wedge (i, j) \in E\}$
- *i is maximal in G if and only if $Suc_G(i) = \emptyset$.*

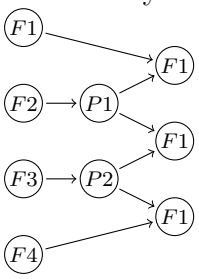
- *i is minimal in G if and only if $Pred_G(i) = \emptyset$.*
- *i is greatest if and only if i is the only maximal job*
- *i is smallest if and only if i is the only minimal job.*
- *i, j are comparable $i \sim_G j$ if either $i < j$ or $j < i$.*
- *i, j are incomparable $i \|_G j$ if they are not comparable.*

Example 1 : Bridge Construction Project
Consider Foundations F_i, Piers P_i, Superstructures S_i.
We get the set of jobs $V = \{F_1, F_2, F_3, F_4, P_1, P_2, S_1, S_2, S_3\}$ with the natural precedence constraints:



$$F_1 < S_1$$
$$F_2 < P_1 < S_1 \qquad P_1 < S_2$$
$$F_3 < P_2 < S_2 \qquad P_2 < S_3$$
$$F_4 < S_3$$

We want to represent this as an activity on node graph for now. This leads to the following directed acyclic graph.



V = set of jobs
$E = \{(i, j) \mid i < j \quad \nexists k : i < k < j\}$
Meaning the transitive reduction of the partial order $<$

$\diamondsuit$

Lemma 1.4:
Let $(V, <)$ be a finite partial order and (V, E) be its transitive reduction. Then E is the smallest (inclusion wise) binary relation on V such that

$$E^{trans} = <$$

where E^{trans} denotes the transitive closure, meaning the smallest transitive relation containing E, e.g. the set of all (i, j) such that there is a finite sequence $i = i_0, \ldots, i_k = j$ with $(i_v, i_{v+1}) \in E$.

Sketch of Proof. By the definition of E we need all pairs $(i, j) \in E$ in any D with $D^{\text{trans}} =<$. Therefore E must be contained in the smallest set D with $D^{\text{trans}} =<$. Let $i < j$ and $(i, j) \notin E$ by definition of E there is a k such that $i < k < j$. By iteration we get the finite sequence in E because $<$ is acyclic. $\square$

Remark:
This lemma does not hold for relations with cycles or infinite ground sets. $\triangle$

Exercise

(1.1) Formulate an algorithm for constructing the transitive closure of a digraph (not necessarily acyclic).

(1.2) Formulate an algorithm for constructing the transitive reduction of a directed acyclic graph (or partial order).

(1.3) Prove $O(n^3)$ run time bound for your algorithms.

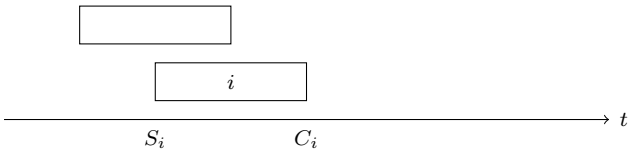
In mathematics predecessor relations are sometimes modeled in a so called Hasse diagram. Where as in computer science and operations research directed acyclic graphs are used.

2 The Deterministic Project Scheduling Model

Every job $j \in V$ has a fixed processing time $x_j \in \mathbb{R}_>$. This gives a processing time vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n_>$. We consider the case with no preemption (no interruption of jobs). So jobs are the smallest units of a project. The precedence constraints are given by a partial order G or $<$ (used synonymously).

Definition 2.1:
*A **schedule** S is a vector $S = (S_1, \ldots, S_n)$ of start times for the jobs. S respects G if and only if $i < j$ implies $S_i + x_i \leq S_j$. Furthermore $C_i = S_i + x_i$ is the **completion time** of job i.*

We may represent a schedule via a so called Gantt Chart



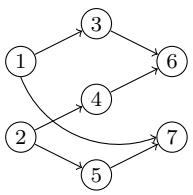
Now we try to model the resource constraints in a mathematically and maybe not that intuitive way.

Definition 2.2:
*The system $\mathcal{F} = \{F_1, \ldots, F_k\}$ of **forbidden sets** consists of antichains F_i of G. F_i is an antichain if $u \parallel_G v$ for all $u, v \in F_i$. These are the smallest sets that must not be scheduled simultaneously at any moment during project execution, but every proper subset can.*
A schedule S repects $\mathcal{F}$ if and only if for every F_i for every t

$$M(t) = \{j \in V \mid S_j < t < C_j\} \not\supseteq F_i$$

Lemma 2.3:
Resource constraints given by forbidden sets model precisely constant resource requirements (constant amount during processing of a job) and availabilities (constant availability during project execution).

Example 2 :



j	1	2	3	4	5	6	7
$r_1(j)$	2	1	1				
$r_2(j)$		1	1	1	2	2	2

$r_i(j)$ is the constant amount of resource i required by job j.

Availabilities: $R_1 = 2$ units of resource 1 and $R_2 = 3$ units of resource 2. This gives the system of forbidden sets

$$\mathcal{F} = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7\}, \{3, 4, 7\}, \{5, 6\}\}$$

Every system $\mathcal{F}$ (with no $F_i \subset F_j$) of antichains of G can be obtained in this way, even with $r_i(j) \in \{0, 1\}$.

j	1	2	3	4	5	6	7	
$F_1 = r_1$	1	1						$R_1 = 1$
$F_2 = r_2$			1	1	1			$R_2 = 2$
$F_3 = r_3$			1	1			1	$R_3 = 1$
$F_4 = r_4$					1	1		$R_4 = 2$
$F_5 = r_5$						1	1	$R_5 = 1$

♢

Example 3 : m-machine problem
We consider m parallel identical machines($R_1 = m$).

j	1	2	...	n
$r_1(j)$	1	1	...	1

Then $\mathcal{F}$ is the set of all $(m + 1)$ element antichains of G. ♢

Remark:

(a) $|\mathcal{F}|$ can be exponential in n.

(b) Often it suffices that $\mathcal{F}$ is given implicitly (by m machines).

(c) A schedule S is feasible for $G, x, \mathcal{F}$ if S respects $G, \mathcal{F}$.

To be able to differentiate between feasible schedules we introduce a regular measure of performance (cost function).

Definition 2.4:
A function $\kappa : \mathbb{R}^n \to \mathbb{R}$ that is non-decreasing in every component describes the cost of performing the project according to schedule S.

$$\kappa(C_1, \ldots, C_n) = \kappa(S, x)$$

*The **cost function** may look like*

(a) $\kappa(C_1, \ldots, C_n) = \max\{C_1, \ldots, C_n\}$ *makespan, project duration*

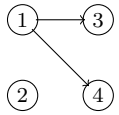
(b) $\kappa(C_1, \ldots, C_n) = \sum_{j=1}^{n} C_j$ *sum of completion times*

(c) $\kappa(C_1, \ldots, C_n) = \sum_{j=1}^{n} w_j C_j$ *weighted sum of completion times*

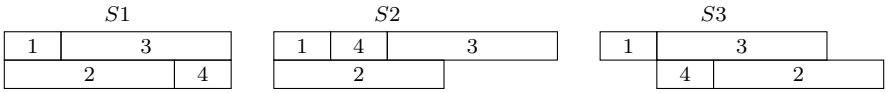
(d) $\kappa(C_1, \ldots, C_n) = \sum_{j=1}^{n} w_j T_j$ *where* $T_j = \max\{0, C_j - d_j\}$ *is the tardiness.*

△

Example 4 : 2 machine model



With $\mathcal{F} = \{\{2, 3, 4\}\}$ and $x = (1, 3, 3, 1)$. There are several possible schedules. (With most left shifted jobs).



S_1 is optimal for (a), S_2 is optimal for (b) and all are feasible. ♢

The optimal schedule depends on the objective function. We may restrict our considerations to left shifted schedules since κ is non-decreasing.
Let us now consider the special case with no resource constraints. Is there a best schedule for G, x? Yes.

Definition 2.5:
We define the ***Early Start*** *(or Earliest Start) schedule*

$$ES_G[x](j) := \begin{cases} 0 & j \text{ is minimal in } G \\ \max\limits_{(i,j)\in E}\{ES_G[x](i) + x_i\} & \text{otherwise} \end{cases}$$

Lemma 2.6:
(a) $ES_G[x]$ *is a schedule that respects* G.

(b) $ES_G[x] \leq S$ *(componentwise) for every feasible schedule* S *respecting* G.

(c) $ES_G[x](j)$ *is the length of a longest chain in* $G|Pred(j)$.

$$ES_G[x](j) = \max\left\{\sum_{i\in U} x_i \,|\, U \text{ is an } \subseteq\text{-max chain in } G|Pred(j)\right\}$$

(d) $ES_G[\cdot] : \mathbb{R}^n_> \to \mathbb{R}^n_\geq$ *is positively homogeneous, convex, monotone, sublinear and continuous.*

Remark:
A chain U is a set of jobs such that $x \sim_G y$ for all $x, y \in U$ holds. The set $G|\text{Pred}(j)$ is the induced subgraph. △

Proof.
(c) Induction on $|\text{Pred}(j)|$

Ind. Start $|\text{Pred}| = 0 \Rightarrow \text{Pred}(j) = \emptyset$ so we maximize over the empty set. Therefore j is minimal in G which implies $ES_G[x](j) = 0$.

Ind. Step $|\text{Pred}(j)| > 0$
$ES_G[x](j) = \max\{ES_G[x](i) + x_i\}$ Thus with $|\text{Pred}(i)| < |\text{Pred}(j)|$ we can use the inductive assumption on i and therefore we get

$$\begin{aligned} ES_G[x](j) &= \max_{(i,j)\in E}\left(\left(\max_U \sum_{k\in U} x_k\right) + x_i\right) \\ &= \max_{U'} \sum_{k\in U'} x_j \end{aligned}$$

Every chain (U') ending in j is a chain (U) in $\text{Pred}(i)$ plus an arc (i, j) for some $i \in \text{ImPred}(j)$.

(b) follows from (c) since the length of a chain in $\mathrm{Pred}(j)$ is a lower bound for S_j for every schedule S.

(a) follows from the definition of ES, but also from (c). Add a dummy job before all minimal jobs. Then l_i is the length of a longest chain in $\mathrm{Pred}(i)$. Thus

$$l_j \geq l_i + x_i \overset{(c)}{\Leftrightarrow} ES_G[x](j) \geq ES_G[x](i) + x_i$$

(d) $ES_G[x](j) = \max_{C\text{chain}} \sum_{i \in C} x_i$ is the maximum of linear functions and therefore inherits the claimed properties.

□

Consequences

(a) Given κ, the minimum cost of planning according to $ES_G[x]$ is

$$\kappa(ES_G[x], x) =: \kappa^G(x) \ .$$

(b) We can consider $\kappa^G(\cdot) : \mathbb{R}^n_> \to \mathbb{R}^1$ as a cost or performance function.

(c) Special case $\kappa = C_{\max}$. Then $C^G_{\max}(x)$ is the length of the longest chain in G.

Remark:

- (c) leads to the CPM (critical path method).
- If the world is simple (no resource constraints) then the early bird rule is optimal.

△

Exercise

(2.1) Prove that if S minimizes $L_{\max}$ (lateness $L_j = C_j - d_j$) then S minimizes $T_{\max}$ (tardiness).

(2.2) The objectives $\sum w_j C_j$ and $\sum w_j L_j$ are equivalent.

(2.3) Describe the makespan polytope

$$P = \left\{ x \in \mathbb{R}^n_\geq \,\middle|\, C^G_{\max}(x) \leq t \right\}$$

of a partial order G by its vertices.

$$t \geq C^G_{\max}(x) \Leftrightarrow \sum_{i \in U} x_i \leq t$$

Hint: Use antichains of G.

3 The Stochastic Project Scheduling Model

We now consider random processing times X_j instead of fixed x_j. Assume that we know the joint distribution P of $X = (X_1, \ldots, X_n)$. The X_i will be independent in most cases but not always.
The structure $[G, P]$ is called a **stochastic project network**. For a measurable (will always be assumed) cost function κ the distribution P_{κ^G} of $\kappa^G(\cdot)$ is well defined. Remember that $\kappa^G(x) = \kappa(ES_G[x] + x)$ is measurable.

Theorem 3.1: Underestimation Error
Let $[G, P]$ be a stochastic project network. Let $\mathbb{E}(X) := (\mathbb{E}(X_1), \ldots, \mathbb{E}(X_n))$ be the vector of average job processing times. If κ is convex, then we have the following comparison between the deterministic planning according to averages (left) and the real expected costs (right)

$$\kappa^G(\mathbb{E}(x)) \leq \mathbb{E}(\kappa^G(\cdot))$$

Proof.

$$\kappa^G(x) = \kappa(ES_G[x] + x) =: f(x_1, \ldots, x_n) \qquad \text{is convex}$$

Thus the theorem follows from Jensen's inequality for convex functions

$$f(\mathbb{E}(X_1), \ldots, \mathbb{E}(X_n)) \leq \mathbb{E}(f(X_1, \ldots, X_n))$$

□

Most of our cost functions are convex (as $C_{\max}$, $T_{\max}$ etc.).

Elementary Proof for $C_{\max}$.
Let $C := \{C^1, \ldots, C^m\}$ be the set of $\subseteq$-maximal chains of G. For C^i define

$$Y^i := \sum_{j \in C^i} X_j$$

as the (random variable) length of chain C^i. Then

$$C^G_{\max}(X) = \max_i Y^i$$

and thus

$$\begin{aligned} C^G_{\max}(\mathbb{E}(X)) &= \max_i \sum_{j \in C^i} \mathbb{E}(X_j) \\ &= \max_i \mathbb{E}\left(\sum_{j \in C^i} X_j\right) \\ &= \max_i \mathbb{E}(Y^i) \\ &= \mathbb{E}(Y^{i_0}) \\ &\leq \mathbb{E}(\max_i Y^i) = \mathbb{E}(C^G_{\max}(X)) \end{aligned}$$

We assume that the expected maximum is attained for some Y^{i_0}. □

Remark:

- Equality holds for $C_{\max}$ if and only if one of the chains is the longest with probability 1.
- The underestimation error can get arbitrarily large (if n grows or the variance of the X_j grows).

$\triangle$

Example 5 : n grows
We have n parallel jobs. G_n is an n element anti-chain. Assume that every X_j is independently exponentially distributed with parameter $\lambda = 1$. Thus $\mathbb{E}(X_j) = \frac{1}{\lambda}$. Then with the memoryless property of the exponential distribution follows

$$\mathbb{E}(C_{\max}^{G_n}) = \mathbb{E}(\text{first completion}) + \mathbb{E}(C_{\max}^{G_{n-1}})$$

Since the minimum of $X_1, \ldots, X_n$ is exponentially distributed with $(\lambda_1 + \ldots + \lambda_n)^{-1}$ we get a recursive formula and can conclude:

$$\frac{1}{\lambda_1 + \ldots + \lambda_n} + \mathbb{E}(C_{\max}^{G_{n-1}}) = \mathbb{E}(C_{\max}^{G_n})$$
$$\frac{1}{n} + \mathbb{E}(C_{\max}^{G_{n-1}}) = \sum_{i=1}^{n} \frac{1}{i} \approx \log n + \gamma$$

where γ denotes the Eulerian constant.
Since $C_{\max}^{G_n}(\mathbb{E}(X)) = 1$, the absolute and relative error may get arbitrarily large. $\diamondsuit$

Summary:

- Stochastic influences are important and need to be considered.
- This has led to PERT method at NASA. It considers only distributions of Y^{i_0} with $\mathbb{E}(Y^{i_0}) = \max_i \mathbb{E}(Y^i)$ for the makespan.

4 Scheduling with Scarce Resources

Introduction and Complexity

Definition 4.1: The model

$V =$ *set of jobs.* $\{1, \ldots, n\}$

$G =$ *graph* (V, E) *of precedence constraints.*

$\mathcal{F} =$ *system of forbidden sets (resource constraints).* $\{F_1, \ldots, F_k\}$

$\kappa =$ *cost function / regular measure of performance.*

$x =$ *deterministic case.* $(x_1, \ldots, x_n)$

$X =$ *stochastic case.* $(X_1, \ldots, X_n)$

In the deterministic case we want find a schedule S that respects G, x and $\mathcal{F}$ (feasible schedule) such that $\kappa(S, x) = \kappa(C_1, \ldots, C_n)$ is minimized.

Definition 4.2: m-machine problems
We have m identical machines / processors and every job needs one, every machine can process only one job at a time.

$$\Rightarrow \mathcal{F} = \{F \subset V \,|\, |F| = m + 1 \text{ and } F \text{is an antichain of } G\}$$

Theorem 4.3:
Let $m = 1$ and $E_G = \emptyset$ (no precedence constraints). Then the following holds.

(1) Idle time does not pay for any κ. Idle time means that a processor is not busy, but there is a job that could be processed.

(2) For $\kappa = \sum w_j C_j$ ***Smith's rule*** *constructs an optimal schedule:*

- *Sort the jobs such that:* $\frac{x_{i_1}}{w_{i_1}} \leq \frac{x_{i_2}}{w_{i_2}} \leq \ldots \leq \frac{x_{i_n}}{w_{i_n}}$
- *Schedule jobs in that order.*

(3) For $\kappa = L_{\max}$ ***Jackson's rule*** *constructs an optimal schedule*

- *Sort jobs such that:* $d_{i_1} \leq d_{i_2} \leq \ldots \leq d_{i_n}$
- *Schedule the jobs in that order.*

(4) The problem with $\kappa = L_{\max}$ and jobs with non-trivial release dates r_j (i.e. $S_j \geq r_j$) is NP-hard.

(5) The cost function $\sum w_j T_j$ gives again an NP-hard problem.

Proof.
(1) Trivial.

(2) A simple exchange argument for two adjacent jobs.

(3) Exercise!

(4) Without proof.

(5) Without proof.

□

The theorem above shows that already small changes may turn a problem from polynomially solvable to NP-hard. This behavior is typical for scheduling. From 1975 to 1985 many scheduling problems have been classified (about 8000) and about 80 per cent of them are NP-hard. [1]

Other examples:$(C^G_{\max})(\mathbb{1})$

We now want to consider the m-machine problem with $m \geq 2$, arbitrary precedence constraints, $x_j = 1$ for all j and $\kappa = C_{\max}$. $x_j = 1$ models jobs in a parallel processor environment where $x_j = 1$ corresponds to a time slot for processing and every long job is subdivided into a chain of pieces with unit length.

Claim:
The case $m = 2$ is polynomially solvable.

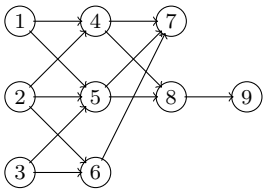
Definition 4.4:
The incomparability graph Incomp(G) of G is given by the vertex set V of G and (i, j) is an edge in Incomp(G) if and only if $i \neq j$ and $i \parallel_G j$.

Theorem 4.5: Fujii et al 67,71
$(C^G_{\max})^{OPT}(\mathbb{1})$ *is equal to the maximum size of a matching in Incomp(G) plus the number of unmatched jobs and one can construct the schedule from the matching in polynomial time.*

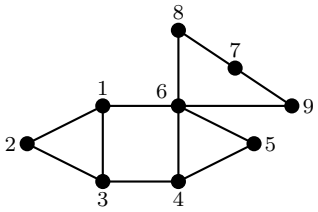
We will illustrate this idea with the following example.

Example 6 :

G



Incomp(G)

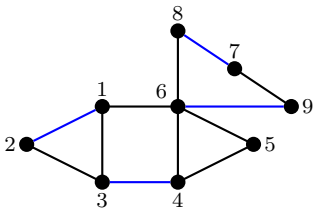


An optimal schedule is given by

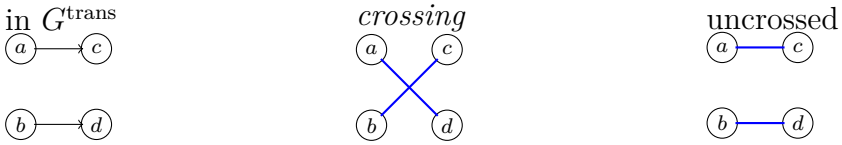
1	3	5	6	7
2	4		8	9

See that every time slot with two jobs in parallel defines a matched edge in Incomp(G). Now from the matching to a schedule.

[1] You can find this and more detailed information on the classification of scheduling problems on: http://www.informatik.uni-osnabrueck.de/Kunst/class



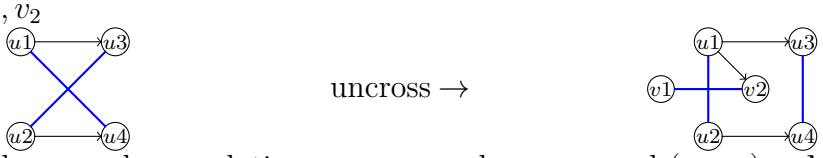
This is a maximum matching, but it cannot be turned into a schedule such that matched jobs are scheduled in the same time slot. The following *crossing* (matching edges cross with respect to precedence constraints) leads to a contradiction.



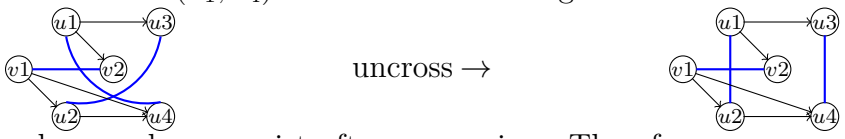
There is no ordering of the time slots respecting the precedence constraints. We want to *uncross* the matching. ♢

Lemma 4.6:
For every matching M, there is a non-crossing matching of the same size that can be obtained by uncrossing crossings in any order.
So every uncrossing reduces the number of crossings.

Proof. Suppose this is not the case. Then the uncrossing creates a new crossing with u_1, v_1, u_2, v_2



This implies the precedence relations $u_1 < v_1$ and $v_2 < u_2$ and $(v_1, v_2) \in M$. Therefore (v_1, v_2) crosses with (u_1, u_4) before the uncrossing



This crossing does no longer exist after uncrossing. Thus for every new crossing, another old crossing involving the same edges is also uncrossed and the number of total crossings drops. □

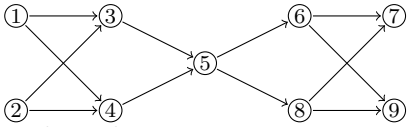
Example 6 : continued
Now assume that the matching M $(= \{(1,2),(3,4),(6,8),(7,9)\}$ e.g.) is non-crossing. We want to construct a schedule from M. First we order the matching edges $(a,b) < (c,d)$ if and only if

$$a < c \vee a < d \vee b < c \vee b < d$$

This defines a unique ordering of the matching edges because there are no crossings. To insert unmatched jobs into this linear order we define $a < (b,c)$(matching edge)

if and only if $a < b \vee a < c$. This is well defined (else we would get a contradiction). Now take a linear extension of that ordering (a linear order that preserves the ordering relations). The linear extension of matching edges and unmatched jobs defines a schedule.

G^*



ES_{G^*}

1	3	5	6	7
2	4		8	9

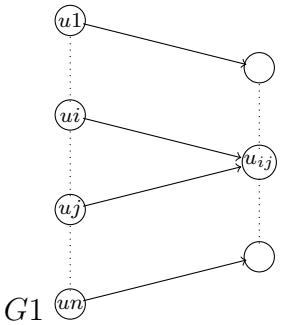
So as claimed in theorem 4.5 every matching gives a schedule S of length $|M| + \#$ unmatched and vice versa. ♢

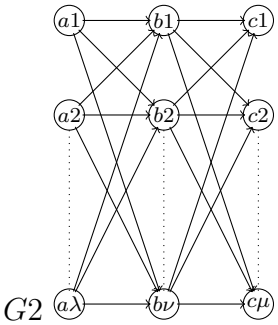
The complexity of the m-machine problem is still open for $m = 3$. If m is part of the input we can use the following theorem.

Theorem 4.7:
The following problem is NP-complete:
For given G, m, t and $x = \mathbb{1}$ is there a feasible schedule with makespan less or equal to t.

Proof. CLIQUE is reducible to the m-machine problem. The CLIQUE problem states: For $G = (V, E)$ and $k \in \mathbb{N}$, does G have a clique of size k? We want to construct an instance of m-machine problem from this instance. The graph G defines vertex jobs (u_i) and edge jobs (u_{ij}). Where every vertex job is schedule before the edge job containing that vertex

$$(i, j) \in E \Rightarrow u_i < u_{ij} \text{ and } u_j < u_{ij}$$





In addition there are dummy jobs that define a frame in which the real jobs must be scheduled together with the time bound t for the makespan. Every $a_i < b_j$ and every $b_i < c_j$. With the number of machines

$$m = \max\left\{k, \binom{k}{2} + n - k, |E| - \binom{k}{2}\right\} + 1$$

define

$$\lambda = m - k \qquad \nu = m - \binom{k}{2} - n + k \qquad \mu = m - |E| + \binom{k}{2}$$

such that $\min\{\lambda\,,\,\nu\,,\,\mu\} = 1$. Set $t = 3$ and $G^* = G_1 + G_2$ (disjoint union).

Claim:
The graph G has a clique of size k if and only if there is a G^* feasible schedule of length at most t.

$\Rightarrow$ Schedule k jobs from the clique in time slot 1. Then we can schedule $\binom{k}{2}$ edge jobs from a k clique and the remaining vertex jobs in time slot 2. Finally schedule the remaining edge jobs in time slot 3.

$\Leftarrow$ Assume that the graph has no clique of size k therefore there are at most $\binom{k}{2} - 1$ in slot 2 and one slot remains empty at time 2. Thus the schedule has length greater than 3.

3	c_1			c_μ	u_{12}	.....
2	b_1	b_ν	u_1	u_{24}	u_{25}	.....
1	a_1		a_λ	u_2	u_4	.....

□

Exercise

(4.1) Show that Jackson's rule constructs an optimal 1-machine schedule for $L_{\max}$ (no precedence constraints).

(4.2) Show that the 3-machine problem can be solved in polynomial time for precedence constraints of fixed width (i.e. the size of the largest antichain is bounded by a constant).

5 Scheduling Policies

Scheduling problems are *highly* NP-hard. Therefore we need polynomial time (approximation) methods to study these problems with varying processing times but fixed $G, \mathcal{F}$, sometimes also independent of κ.

Definition 5.1:
*A **planning rule** for $[G, \mathcal{F}]$ is a function $\Pi : \mathbb{R}^n_> \to \mathbb{R}^n_\geq$ that assigns to each vector x of processing times a schedule $\Pi[x]$ that respects $G, \mathcal{F}$ and x.*

We can now speak of monotone, continuous, etc. (properties of) planning rules.
For fixed κ and a planning rule Π the function:

$$\kappa^\Pi : \mathbb{R}^n_> \to \mathbb{R}^1 \quad \text{with} \quad \kappa^\Pi[x] = \kappa(\Pi(x) + x)$$

gives the performance cost resulting from planning according to Π and x. For a fixed class $\mathcal{P}$ of planning rules

$$\rho^\mathcal{P}(\kappa, x) := \inf\{\kappa^\Pi(x) \,|\, \Pi \in \mathcal{P}\}$$

denotes the optimal value over $\mathcal{P}$ for fixed x.
If processing times are random, we want to find a planning rule that is best on average, i.e.

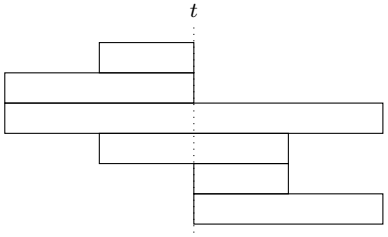
$$\rho^\mathcal{P}(\kappa) := \inf\{\mathbb{E}[\kappa^\Pi] \,|\, \Pi \in \mathcal{P}\}$$

Dynamic Representation of Planning Rules and Ploicies

This is a different implicit representation.
Decision Times

- Project start $t = 0$.
- Completion of a job.
- Times at which information becomes available.

These are the times, when we have to make a decision.
Consider the situation at decision time t:



- Set of completed jobs $C(t)$.
- Set of busy jobs $B(t)$.
- Set of jobs started at t $S(t)$.

What constitutes a decision:

- $S(t)$ The set of start jobs
- t^{plan} The next tentative decision time.
- $t^{next} = \min\{t^{plan},$ next completion $\}$

Definition 5.2:
A planning rule is non-anticipative or a ***policy*** *if* t^{plan} *and* $S(t)$ *depend only on the history up to time* t*.*

History hereby means:

- Processing times and starting times of jobs in $C(t)$
- Current processing times and start times of jobs in $B(t)$.
- Time t.
- $G, \mathcal{F}$ etc.

This gives information about conditional distribution obtained from the given joint distribution Q of processing time conditioned on the history.
In dynamic programming language we also say state instead of history and action instead of decision.

Example 7 :

- Smith' rule is not a policy, but Smith's rule based on expected length

$$\frac{\mathbb{E}(x_{i_1})}{w_{i_1}} \leq \ldots \leq \frac{\mathbb{E}(x_{i_n})}{w_{i_n}}$$

  would be a policy.

- The matching algorithm for the 2-machine problem is a policy if we take the ordering of jobs obtained from the matching.

$\diamondsuit$

Example 8 : Non-anticipative is worse than anticipative
Let $V = \{1, 2, 3\}$ without precedence constraints and the distribution

$x = (1, 1, 2)$	$y = (1, 2, 1)$	$z = (2, 1, 1)$
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

The best anticipative planning rule does the best for every realization.

x :

1	2
3	

y :

1	3
2	

z :

2	3
1	

Thus we get $\mathbb{E}[C^{\Pi}_{\max}] = 2$.
The best non-anticipative planning rule must make a decision at time $t = 0$ without knowing the future. So without loss of generality $S(0) = \{1, 2\}$ implies the schedules:

x :

1	3
2	

y :

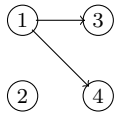
1	3
2	

z :

1	
2	3

Thus we get $\mathbb{E}[C^{\Pi}_{\max}] = \frac{7}{3}$. This is the unavoidable loss due to non-anticipativity. $\diamondsuit$

Example 9 : Tentative Decision Times are Necessary



Consider the 2-machine problem with the forbidden set $F = \{2, 3, 4\}$.

Let the $X_j \sim \exp(a)$ be exponentially distributed and independent with a common due date d. Consider penalties for lateness v for job 2 and w for jobs $3, 4$ with $v \ll w$. We want to minimize the expected penalties.
If we start the jobs 1 and 2 at $t = 0$ we risk that job 2 blocks a machine and we have to do the expensive jobs 3 and 4 sequentially. Starting only job 1 and wait for its completion the deadline may be approaching and we have a short span for the remaining jobs.
Thus start job 1 at time $t = 0$ and fix a tentative decision time t^{panic}. If $C_1 \leq t^{\text{panic}}$ start 3 and 4 at C_1, else start 2 at t^{panic}. ♢

Without loss of generality we may require that $S(t) \neq \emptyset$ at tentative decision times. Otherwise the policy without that decision time would define the same function $\Pi : \mathbb{R}^n_> \to \mathbb{R}^n_\geq$.
Thus a planning rule has a finite number of decision times, especially at most $2n-1$, because at $t = 0$ and every completion except the last one job starts and we have at most $(n-1)$ tentative decision times.

Lemma 5.3:
Let Π *be a policy. Then the history at decision time* t *is completely determined by*

- *The processing time* x_j *of all* $j \in C(t)$.
- *The current processing time* $\bar{x}_j$ *of all* $j \in B(t)$.
- *Time* t.

Meaning, from that information, we can construct S_j, C_j *of the history.*

Proof. By induction along the decision times.

$t^{(1)} = 0$ The policy Π defines $S(t^{(1)})$ uniquely. Thus we know all $S_j = 0$ for all $j \in S(t^{(1)})$.

$t^{(k)} \to t^{(k+1)}$

Case 1 $t^{(k+1)}$ s not a completion time. Since we knew all S_j and C_j at $t^{(k)}$ by the inductive assumption and nothing completes and the start time of new jobs is $t - \bar{x}_j$, we know all S_j and C_j up to $t^{(k+1)}$.

Case 2 $t^{(k+1)}$ is a completion time (first completion after $t^{(k)}$). The start times of new jobs is $t - \bar{x}_j$ and the completion times of newly completed jobs is t.

□

The lemma above implies the following theorem.

Theorem 5.4:
A planning rule Π *is non-anticipative if and only if the following condition holds:*

$$x \sim_t y \text{ and } \Pi[x](j) = t \Rightarrow \Pi[y](j) = t \tag{NA}$$

We say that x, y *look the same to* Π *at time* t $(x \sim_t y)$ *if and only if*

$$\begin{aligned} x_j = y_j \quad & \forall j \in C(t) = C_x(t) = C_y(t) \\ \bar{x}_j = \bar{y}_j \quad & \forall j \in B(t) = B_x(t) = B_y(t) \end{aligned}$$

The above theorem is a condition in the interpretation of Π as a function. The relation $\sim_t$ constitutes an equivalence relation E_t on $\mathbb{R}^n_>$. So $\Pi : \mathbb{R}^n_> \to \mathbb{R}^n_\geq$ is a policy if and only if $\Pi[x]$ is a feasible schedule and for all x and Π fulfills (NA).

Theorem 5.5:

(a) $E_0 = \mathbb{R}^n_> \times \mathbb{R}^n_>$, *i.e.,* $x \sim_{t=0} y$ *for all* x, y

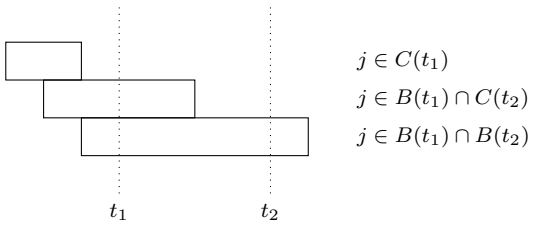
(b) $t_1 < t_2$ *implies* $E_{t_1} \supset E_{t_2}$ *(i.e.* $x \sim_{t_2} y \Rightarrow x \sim_{t_1} y$*)*

(c) $E_\infty = \{(x, x) \,|\, x \in \mathbb{R}^n_>\}$ *(i.e.* $x \sim_\infty y \Rightarrow x = y$*)*

Proof.
(a) Clear since $C(0) = \emptyset$ and $\bar{x}_j = \bar{y}_j = 0$ for all j.

(b) Now $x \sim_{t_2} y$ implies

$$x_j = y_j \quad \forall j \in C(t_2) \qquad \bar{x}_j = \bar{y}_j \quad \forall j \in B(t_2)$$



Now for $j \in C(t_1)$ we get $j \in C(t_2)$ and with $x \sim_{t_2} y$ follows $x_j = y_j$ at t_1.
For $j \in B(t_1)$ implies two cases $j \in B(t_2)$ or $j \in C(t_2)$.

Case 1 $j \in B(t_2)$ implies $\bar{x}_j = \bar{y}_j$ at t_2 and thus $\bar{x}_j = \bar{y}_j$ at t_1.

Case 2 $j \in C(t_2)$. Use lemma **??** at t_2. Thus S_j and C_j are the same for x, y and therefore $\bar{x}_j = \bar{y}_j$ at time t_1.

(c) Let $x \neq y$ say $x_j \neq y_j$. For $t \to \infty$ Π will see the difference in the history.

□

Definition 5.6: Properties of Planning Rules

- Π_1 ***dominates*** Π_2 *if and only if* $\Pi_1(x) \leq \Pi_2(x)$ *(component-wise) for all* x.

- Π *is* ***minimal*** *in a class* $\mathcal{P}$ *of planning rules if*
  (i) $\Pi \in \mathcal{P}$
  (ii) $\Pi' \in \mathcal{P}$ *and* $\Pi' \leq \Pi$ *implies* $\Pi' = \Pi$.
- Π *is* ***elementary*** *if and only if* Π *starts jobs only at completion of other jobs.*

To use approximate methods under data deficiencies we require **stability**:

$$\begin{matrix} Q_n \text{ approximates } Q \\ \kappa_n \text{ approximates } \kappa \end{matrix} \Rightarrow OPT(Q_n, \kappa_n) \text{ approximates } OPT(Q, \kappa)$$

Where Q_n approximates Q ($Q_n \rightharpoonup Q$) means weak convergence of measures[2] and κ_n approximates κ means uniform convergence.
In general we have no stability for optimal policies.

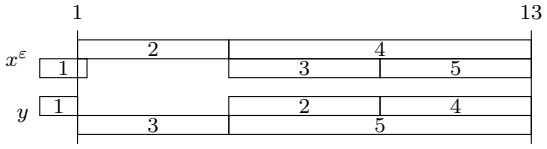
Example 10 : Excessive use of information yields instability

① ②⟶④ ③⟶⑤

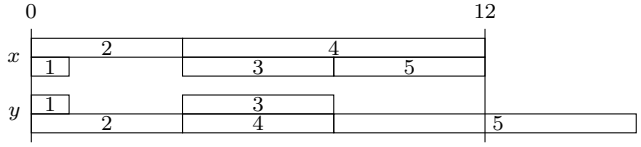
Consider the objective $\min \mathbb{E}(C_{\max})$, the only forbidden set $F = \{2, 3\}$ and the joint distribution:

$$Q^\varepsilon \; : \; \begin{cases} x^\varepsilon = (1+\varepsilon, 4, 4, 8, 4) & \text{with probability } \frac{1}{2} \\ y = (1, 4, 4, 4, 8) & \text{with probability } \frac{1}{2} \end{cases}$$

If we start job 1 first and wait till $t = 1$ we know which realization we have and can plan the rest optimal.



This gives $\mathbb{E}_{Q^\varepsilon}(C_{\max}) = 13$.
Now $\varepsilon \to 0$ implies $Q^\varepsilon \to Q$. Therefore we gain no information at $t = 1$ and thus start job 2 (or 3) at $t = 0$.



Which leads to $\mathbb{E}_Q(C_{\max}) = 14 \neq 13 = \lim \mathbb{E}_{Q^\varepsilon}(C_{\max})$. ♢

So we want to only rely on robust information at time t this includes: which jobs have completed and which jobs are running at t. If we start jobs only at completion of other jobs (elementary) we may hope for stability.

[2] $Q_n \rightharpoonup Q \Leftrightarrow \int f \, dQ_n \to \int f \, dQ \quad \forall f$ continuous and bounded.

Definition 5.7:
*A class $\mathcal{P}$ of policies is called **stable** if for every sequence $Q^j \rightharpoonup Q$, every countable subset $\mathcal{P}'$ of $\mathcal{P}$ and every continuous κ the following holds:*

$$\lim_{j\to\infty} \rho^{\mathcal{P}'}(\kappa, Q^j) = \rho^{\mathcal{P}'}(\kappa, Q)$$

Theorem 5.8:
Let $\mathcal{P}$ be a stable class of policies. Then

(1) Every $\Pi \in \mathcal{P}$ is continuous.

(2) If $\mathcal{P}'$ is a finite class of continuous policies then $\mathcal{P} \cup \mathcal{P}'$ is stable.

(3) For all $Q^j \rightharpoonup Q$ and $\kappa^j \to \kappa$ the following holds

$$\rho^{\mathcal{P}}(\kappa^j, Q^j) \to \rho^{\mathcal{P}}(\kappa, Q)$$

Proof.
(1) Suppose Π is not continuous. Then the mapping $x \mapsto \Pi[x](j) + (x_j)$ is discontinuous for some job j. Thus there is a sequence $x^k \to x$ with

$$\lim_{k\to\infty} \Pi[x^k](j) + x_j^k \neq \Pi[x](j) + x_j$$

Choose $\kappa(C_1, \ldots, C_n) = C_j$, Q^k the point distribution in x^k converging to Q the point distribution in x and $\mathcal{P}' = \{\Pi\}$. This implies

$$\begin{aligned}\lim_{k\to\infty} \rho^{\mathcal{P}'}(\kappa, Q^k) &= \lim_{k\to\infty} \Pi[x^k](j) + x^k(j) \\ &\neq \Pi[x](j) + x_j \\ &= \rho^{\mathcal{P}'}(\kappa, Q)\end{aligned}$$

Thus the weak stability is not satisfied.

(2) Show that stability holds by adding one continuous policy at a time.

(3) This is more difficult and needs results on convergence of probability measures. (Especially sufficient conditions.)

□

Exercises

(5.1) Find an example for $\sum_j w_j C_j$ with independent processing times in which tentative decision times lead to better policies. Can you do it without precedence constraints?

(5.2) Try the same for $C_{\max}$.

(5.3) Show that the class of elementary policies is unstable.

There are special classes of policies which can be approached from three different point of views.

6 Priority Policies

The combinatorial objects corresponding to these policies are priority lists.

Definition 6.1:
A planning rule Π *is a priority planning rule or* ***priority rule*** *for* $[G, \mathcal{F}]$ *if*

1. Π *is elementary.*
2. *At every decision time* t*, there is a priority list on the set of still unscheduled jobs:*
$$L(t) = j_1 < j_2 < \ldots < j_k$$
3. Π *considers jobs* j *in* $L(t)$ *one by one in the order of* $L(t)$ *and sets* $S_j = t$ *if possible with respect to* $G, \mathcal{F}$ *and the previously started jobs.*

Π is **static** if every $L(t)$ is a sublist of $L(0)$. Otherwise Π is called **dynamic**.
For priority policies, the list $L(t)$ may only depend on the history up to time t.

Remark:
Smith's rule is a static priority rule but not a priority policy. Smith's rule using expected lengths is a priority policy. △

Advantages of Priority Policies

(1) Every priority policy is minimal among all policies.

(2) Easy to implement

(3) There is an approximation guarantee for some problems. E.g. :
$$C^{\Pi}_{\max}(x) \leq \left(2 - \frac{1}{m}\right) OPT^{C_{\max}}(x)$$
for the m-machine problem with arbitrary processing times.

Proof.
(1) Suppose there is some $\Pi' \leq \Pi$ for a priority policy Π. Consider a fixed x at time $t = 0$. Thus $\Pi[x]$ starts as many jobs as possible in the order of $L(0)$ at $t = 0$. Since $\Pi' \leq \Pi$, $\Pi'[x]$ has the same start set. Now look at the first completion with respect to x. Π' cannot start new jobs earlier since all resources are used. Thus $\Pi[x] = \Pi'[x]$ with the same argument as above. Iterations gives the claim since Π and Π' have to start the same jobs at every completion time.

(2) Clear

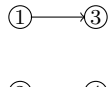
(3) Without proof (later).

□

Disadvantages of Priority Policies

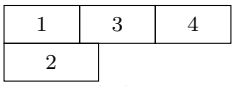
Priority policies are neither continuous nor monotone and thus may cause instability.

Example 11 :

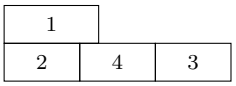


Consider $L = 1 < 2 < 3 < 4$ and the forbidden set $F = \{3, 4\}$. For fixed x_2, x_3, x_4 and $x_1 \in [x_2 - \varepsilon, x_2 + \varepsilon]$. $S_4 = \Pi[\cdot](4)$ is discontinuous and not monotone.

$x_1 < x_2 \Rightarrow$



$x_2 < x_1 \Rightarrow$

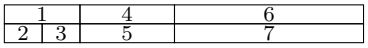


(S_4 may jump while x grows a little). ♢

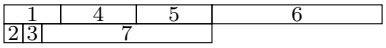
Priority Policies Anomalies (Graham Anomalies)

(a) We want to minimize the makespan on 2 identical machines using the priorities $1 < 2 < 3 < \ldots$. Then $C_{\max}$ may grow though x gets smaller:

$x = (4, 2, 2, 5, 5, 10, 10)$

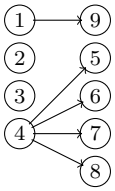


$y = x - 1 = (3, 1, 1, 4, 4, 9, 9)$



In the first case we have $C_{\max} = 19$ and after shortening we get $C_{\max} = 20$.

(b)



We consider $m = 2$, $x = (3, 2, 2, 2, 4, 4, 4, 4, 9)$ and $L = 1 < \ldots < 9$. $C_{\max}$ grows when $4 < 5$ and $4 < 6$ are deleted.

(c) $C_{\max}$ grows when more machines are deleted. Take the previous G but $m = 3$. Then $C_{\max}$ grows when $m = 4$ machines are available.

Remark:
In general, there is no priority policy that yields an optimal schedule for fixed x. To see this take example (a) with processing times y. △

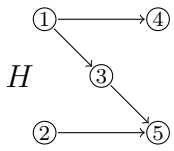
Exercise

(6.1) Show that there are no Graham anomalies for m-machine problems without precedence constraints. Show that, in this case, every static priority policy is continuous and monotone.

7 Early Start Policy

Recall the ES-function induced by a partial order H on V. Then $ES_H[x]$ is the earliest start schedule with respect to H and x.

Example 12 :



This graph H leads to the earliest start schedule:

$$ES_H[x] = (0, 0, x_1, x_1, \max\{x_1 + x_3, x_2\})$$

♢

Thus $ES_H[\cdot]$ is a function with the same domain and range as a policy Π for $[G, \mathcal{F}]$ and $ES_H[\cdot]$ has nice properties (continuous, monotone).
Question: Given $[G, \mathcal{F}]$, when is $ES_H[\cdot]$ a policy for $[G, \mathcal{F}]$?

Theorem 7.1:
$ES_H[\cdot]$ is a planning rule for $[G, \mathcal{F}]$ if and only if

(1) H extends G, i.e., $i <_G j \Rightarrow i <_H j$.

(2) No antichain of H is forbidden, i.e., $F \in \mathcal{F} \Rightarrow F$ is not an antichain of H.

Proof.

$\Rightarrow$ By contradiction.

(1) Assume $i <_G j$ but $i \not<_H j$. Define $x \in \mathbb{R}^n_>$ with $\varepsilon < \frac{1}{n^2}$ by

$$x_k := \begin{cases} \varepsilon & k \neq i \\ 1 & k = 1 \end{cases}$$

and therefore

$$ES_H[x](j) \leq (n-2) \cdot \varepsilon < 1$$

$x_i = 1$ implies that j does not wait for i in the schedule $ES_H[x]$. Thus $ES_H[x]$ does not respect G. Therefore $ES_H[\cdot]$ is not a planning rule for $[G, \mathcal{F}]$.

(2) Suppose $F \in \mathcal{F}$ is an antichain of H. Define $x \in \mathbb{R}^n_>$ with $\varepsilon < \frac{1}{n^2}$ by

$$x_k = \begin{cases} \varepsilon & k \notin F \\ 2 & k \in F \end{cases}$$

Then for $j \in F$ because j has only short predecessors:

$$ES_H[x](j) \leq (n-2)\varepsilon < 1$$

At $t = 1$ all jobs in F are busy. Thus ES_H is not a planning rule for $[G, \mathcal{F}]$

$\Leftarrow$ H extends G therefore $ES_H[\cdot]$ respects G. No antichain of H is forbidden and thus $ES_H[\cdot]$ respects $\mathcal{F}$.

□

Definition 7.2:
*A partial order H on V is called **feasible** for $[G, \mathcal{F}]$ if it respects G (i.e.H extends G) and $\mathcal{F}$ (no antichain of H is in $\mathcal{F}$).*

Theorem 7.3:
$ES_H[\cdot]$ is a planning rule for $[G, \mathcal{F}]$ if and only if H is feasible for $[G, \mathcal{F}]$. In this case $ES_H[\cdot]$ is already a policy, i.e., non-anticipative.

Proof. The first part follows immediately from the previous theorem **??**.
Let $x \sim_t y$ and $ES_H[x](j) = t$ so that x, y have the same history up to time t. Thus for all $i \in \mathrm{Pred}_H(j)$ where $x_i = y_i$. Therefore since ES_j is the longest path length in $\mathrm{Pred}_H(j)$ we get

$$ES_H[y](j) = ES_H[x](j) = t$$

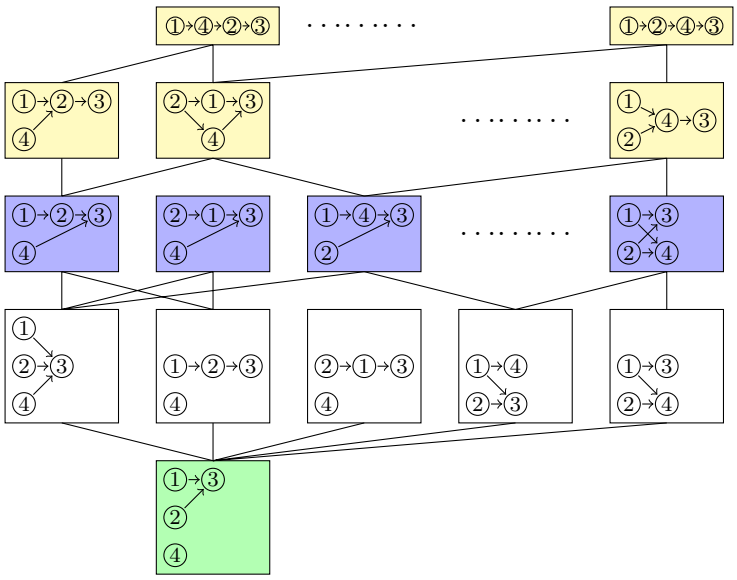
□

Let $\mathcal{E}(G)$ denote the set of all extension of G (including G). $\mathcal{E}(G)$ is a partial order under the order relation:

$$H_1 \prec H_2 \Leftrightarrow H_2 \text{ extends } H_1 \quad \text{i.e. } i <_{H_1} j \Rightarrow i <_{H_2} j$$

We call $\mathcal{E}(G)$ the **extension order** of G.

Example 13 : Conflict Resolution Tree



We consider the forbidden sets $\mathcal{F} = \{\{1, 2, 4\}, \{3, 4\}\}$. In the above diagram we used the coloring
green root blue minimum feasible yellow feasible ♢

Claim:
$\mathcal{E}(G)$ has nice properties.

(1) $H_1 = (V, E_1)$, $H_2 = (V, E_2)$. H_1 is an immediate predecessor of H_2 if and only if $|E_2^{\text{trans}} \backslash E_1^{\text{trans}}| = 1$.

(2) All $\subset$-maximal chains from H_1 to H_2 have the same length.

(3) H_1 and H_2 have a (unique) largest common predecessor $H = (V, E)$ with $E^{\text{trans}} = E_1^{\text{trans}} \cap E_2^{\text{trans}}$.

(4) H_1, H_2 have a (unique) smallest common successor $H = (V, E)$ if and only if $E_1 \cup E_2$ is acyclic. Then $E^{\text{trans}} = (E_1 \cup E_2)^{\text{trans}}$ holds.

Note that (3) and (4) say that $\mathcal{E}(G)$ equipped with an artificial greatest element is a semi-lattice.

Consequences

(1) The set of feasible order is *upwardly closed* in $\mathcal{E}(G)$, i.e., H_1 feasible and $H_1 \prec H_2$ imply H_2 is feasible.

(2) $H_1 \prec H_2$ if and only if $ES_{H_1} \leq ES_{H_2}$

(3) ES_H is a minimal ES-policy if and only if H is minimal feasible in $\mathcal{E}(G)$.

(4) The set of ES-policies for $[G, \mathcal{F}]$ may be identified with the set of feasible orders in $\mathcal{E}(G)$.

ES policies lead to optimal schedules in the deterministic case. This is not the case for priority policies! In the above example with processing times y, every priority policy would start job 7 at the completion of 3, leading to a non-optimal schedule.

Theorem 7.4:
Consider $[G, \mathcal{F}]$. *For every* x *there is an ES-policy* $\Pi = ES_H$ *such that*

$$OPT(\kappa, x) = \kappa^H(x) = \kappa^\Pi(x)$$

Note that Π *and thus* H *depend on* x *and* κ. *This in particular means*

$$OPT(\kappa, x) = \min\{\kappa^H(x) \mid H \text{ feasible } \} = \min\{\kappa^H(x) \mid H \text{ minimal feasible } \}$$

Proof.

$\leq$ Let the minimum of the right handside be attained by H. Then $ES_H[x]$ is a feasible schedule for $[G, \mathcal{F}]$ and x. This implies

$$\kappa(ES_H[x], x) = \kappa^H(x) \geq OPT(\kappa, x)$$

$\geq$ Let S be an optimal schedule for $(G, \mathcal{F}, x, \kappa)$. Consider the partial order $H = H(S, x)$ induced by S and x:

$$i <_H j \quad \text{if} \quad S_i + x_i \leq S_j$$

Claim: 1
A is an antichain of $H(S,x)$ if and only if there is a time t such that all $j \in A$ are busy, i.e., $S_j < t < S_j + x_j$ for all $j \in A$.

Proof. Induction on $|A|$. (Claim 1 corresponds to the Helly property of a set of intervals.) □

Claim: 2
$H(S,x)$ extends G.

Proof. $i <_G j$ implies $S_i + x_i \leq S_j$ since S is feasible. By definition of $H(S,x)$ follows $i <_H j$. □

Claim: 3
No antichain of $H(S,x)$ is forbidden.

Proof. If A is an antichain of $H(S,x)$ then claim (1) implies that A is processed at some time t in (S,x). Since S is feasible, A is not a forbidden set. □

Claim (2) and claim (3) imply that $H(S,x)$ is a feasible order.
S respects $H(S,x)$ by construction and $ES_{H(S,x)}[x]$ is the component-wise best schedule for $H(S,x)$.

$$\Rightarrow \kappa^{H(S,x)}(x) \leq \kappa(S,x)$$

So there is a minimal feasible order $\bar{H} \prec H(S,x)$ with $\kappa^{\bar{H}}(x) \leq \kappa(S,x)$.

□

Remark:

(1) Every schedule S and vector x induce such an order $H(S,x)$ (since $x_i > 0$).

(2) $H(S,x)$ is an interval order, i.e., an order H whose elements j that can be represented by intervals $I_j \subset \mathbb{R}$ such that

$$i <_H j \quad \text{if } I_i \text{ is entirely to the left of } I_j \text{ (except for endpoints)}$$

For $H(S,x)$ set $I_j = [S_j, S_j + x_j]$

(3) Interval orders have nice properties. (See Exercises).

(4) Claim (1) corresponds to the Helly property of a set of intervals: If any two intersect then all intersect.

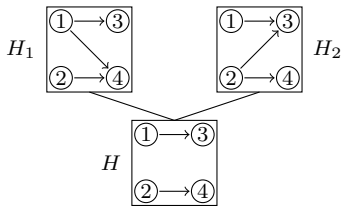
△

Consequences

In the deterministic case we have

$$OPT(\kappa, x) = \min\{\kappa^H(x) \mid H \text{ is a minimal feasible interval order } \}$$

This does not hold in the stochastic case.



For every deterministic x we have

$$ES_H[x] = \min\{ES_{H_1}[x], ES_{H_2}[x]\}$$

Thus one of H_1, H_2 is as good as H.

For random processing times, when we minimize expected makespan, H may be better. For example let every job have a processing time 1 and 3 with probability $\frac{1}{2}$ each, and independent from each other. This gives for example

$$\mathbb{E}(C^{H_1}_{\max}) = \mathbb{E}(C^{H_2}_{\max}) = 4.875 \qquad \mathbb{E}(C^{H}_{\max}) = 4.75$$

Note that H is not an interval order but H_1 and H_2 are.

Theorem 7.5:
Let Π *be a policy for* $[G, \mathcal{F}]$. *Then* Π *is an ES-policy if and only if* Π *is convex.*

Proof. Later. □

Exercises

(7.1) Prove the properties of $\mathcal{E}(G)$

(7.2) (Characterization of Interval Orders)
Show that the following conditions are equivalent:

(1) H is an interval order.

(2) H does not contain $(2+2)$ ○⟶○ ○⟶○ as induced suborder.

(3) There is an ordering $j_1, \ldots, j_n$ of V with

$$\text{Pred}(j_1) \subseteq \ldots \subseteq \text{Pred}(j_n)$$

(4) There is a numbering $A_1, \ldots, A_m$ of the maximal antichains of H such that for every $j \in V$, all A_k containing j occur consecutively in the numbering. (Consecutiveness property of maximal antichains.)

(7.3) The m-machine problem can be solved in polynomial time on interval orders.

(7.4) For every minimal feasible order H, there are κ and Q such that H is the only optimal (minimal) feasible order, meaning:

$$\mathbb{E}_Q[\kappa^H] < \mathbb{E}_Q[\kappa^{H'}] \qquad \text{for all minimal feasible } H' \neq H$$

Does this also holds for $\kappa = C_{\max}$?

8 Constructing ES-Policies

Lemma 8.1:
H is feasible for $[G, \mathcal{F}]$ if and only if

(1) $G \prec H$

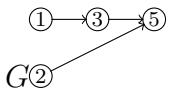
(2) For every $F \in \mathcal{F}$ there are $i_F, j_F \in F$ with $i_F <_H j_F$

For (2) we say F is destroyed by $i_F < j_F$, meaning the resource conflict given by F is settled by letting j_F wait for i_F.

Proof. Obvious, since no $F \in \mathcal{F}$ is an antichain of H. □

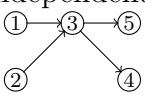
The idea is to solve conflicts on forbidden sets by telling who must wait for whom. This may be contradictory.

Example 14 :



$\mathcal{F} = \{\{2,3\},\{3,4\},\{1,2,4\}\}$

There are 24 independent choices of $i_F < j_F$ but only 9 lead to feasible orders. This is for example $2 < 3$, $3 < 4$ and $1 < 4$.



But $2 < 3$, $3 < 4$ and $1 < 4$ lead to a contradiction because $E_H \cup \{(2,3),(3,4),(4,1)\}$ is not acyclic. ♢

H and a choice of $i_F < j_F$ defines an ES policy if and only if H and the choice is acyclic.
Let us organize the construction in a tree, the conflict settling tree. Let G be the root and extensions of G be the nodes. The children of a node H are all extensions H' obtained by settling the conflict on one (yet unsettled) forbidden set. The leaves correspond to feasible orders. We may use a suitable ordering of the forbidden sets. These orderings determine the tree.

Theorem 8.2:
The conflict settling tree is a suborder (not necessarily induced) of $\mathcal{E}(G)$ containing all minimal feasible orders.

Proof.

- It is obviously a suborder.
- It is not necessarily induced, since all the order relations $H_1 \prec H_2$ in the tree are also present in $\mathcal{E}(G)$. But there may be more if there are several $F \in \mathcal{F}$ with $|F| > 2$.
- All minimal feasible orders are contained:
  Let H be minimal feasible and $F_1, \ldots, F_k$ be the order defining the tree. Now lemma 8.1 gives that for every F_r there is some $i_{F_r} < j_{F_r}$ in H. Thus adding the $i_{F_r} < j_{F_r}$ in the order $1, \ldots, k$ yields H.

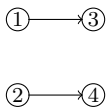
□

Note that not every leaf of the conflict settling tree is minimal feasible.

Remark:
The conflict settling tree may be used for Branch & Bound algorithms for calculating an optimal schedule for given κ and x. The B & B algorithms can be combined with dynamic decision models (Branch only at decision times). In this way only a subset of all forbidden sets will be considered.
In the stochastic case, the optimal values obtained by priority / ES-policies are incomparable. (Remember: In the deterministic case, ES is better.) △

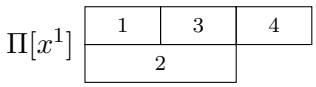
Example 15 :

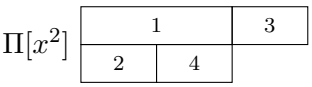
(1) $OPT^{\text{Prior}} < OPT^{ES}$



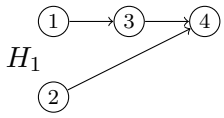
$\mathcal{F} = \{\{3,4\}\}$ $\kappa = C_{\max}$ with the realisations $x^1 = (1,2,0,1)$ and $x^2 = (2,1,1,1)$. Each has probability 0.5

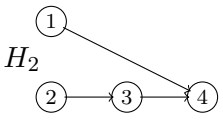
Now the priority policy leads to:





and thus $\mathbb{E}[\kappa^{\Pi}] = 3$. The minimal feasible orders are





and thus $\mathbb{E}[\kappa^{H_1}] = \mathbb{E}[\kappa^{H_2}] = 3.5$.

(2) $OPT^{ES} < OPT^{\text{Prior}}$ holds already in the deterministic case.

♢

Exercises

(8.1) Construct an example in which the conflict settling tree contains leaves that are not minimal feasible.

(8.2) What is the complexity of adding a precedence constraint $i_F < j_F$ to an order H? What is a good data structure for the orders in the conflict settling tree to allow fast updating with respect to adding precedence constraints.

9 Preselective Policies

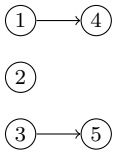
We want to generalize the ES-planning rules by relaxing the *rule* for solving the conflict on a forbidden set f.

Definition 9.1: ES-policies
*For every $F \in \mathcal{F}$ choose $i_F, j_F \in F$ and add $i_F < j_F$ to G, i.e. choose a **waiting job** j_F and a job i_F for which j_F must wait.*

Definition 9.2: Preselective planning rules
For every $F \in \mathcal{F}$, choose a waiting job $j_F \in F$ that must wait for any job in F, i.e. j_F can start after the first job in $F \backslash \{j_F\}$ completes.

The sequence $s = (j_{F_1}, \ldots, j_{F_k})$ of waiting jobs $j_{F_r} \in F_r$ for $\mathcal{F} = \{F_1, \ldots, F_k\}$ is called a **selection** for $\mathcal{F}$.

Example 16 :



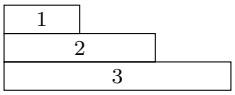
The forbidden sets

$$\mathcal{F} := \{\{1,5\},\{2,3,4\},\{2,4,5\}\}$$

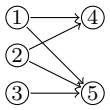
defines a selection $s = (5,4,4)$.

What are the possible schedules, when we do early start scheduling with respect to G and the waiting conditions coming from the selection s?

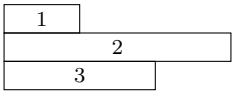
(1) $x_1 < x_2 < x_3$ leads to ES of a feasible order.



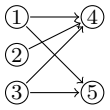
H_1



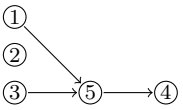
(2) $x_1 < x_3 < x_2$ so 4 waits for 2 or 5



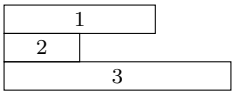
H_{21}



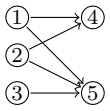
H_{22}



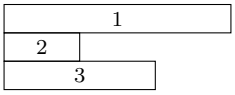
(3) $x_2 < x_1 < x_3$



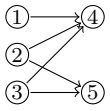
H_3



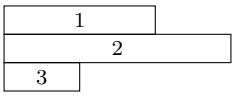
(4) $x_2 < x_3 < x_1$



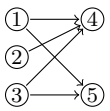
H_4



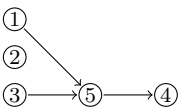
(5) $x_3 < x_1 < x_2$ so 4 waits for 2 or 5



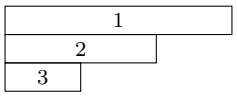
H_{51}

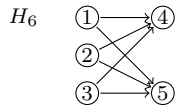


H_{52}



(6) $x_3 < x_2 < x_1$





All other cases are subsumed by (1) to (6) by equivalence. Further we can see:

$$H_1 = H_3 \qquad H_{51} = H_{21} \qquad H_{52} = H_{22} \qquad H_1 \prec H_6$$

♢

This example shows that depending on x we obtain a different feasible order H with $\Pi[x] = ES_H[x]$ (holds for every elementary policy). Furthermore we get

$$\Pi = \min\{ES_H \mid H \text{ induced by } \Pi[x] \text{ for some } x\}$$

This does not hold for every elementary policy.

Questions

(1) When is a selection contradictory? Can this be easily checked?

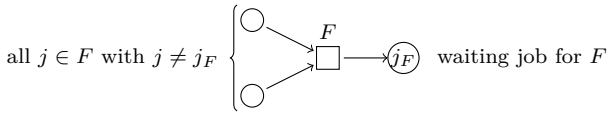
(2) Does a feasible selection define a policy? I.e. is a preselective policy (non-anticipative)?

(3) What is the relationship with ES-policies? Is $\Pi = \min\{ES_H \mid \ldots\}$ for some set of feasible ES-policies?

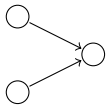
(4) How to calculate the start times of a preselective policy?

(5) Do preselective policies still have nice properties?

The combinatorial representation of preselective planning rules are so called AND/OR networks.
To see this, consider a selected job j_F for $F \in \mathcal{F}$. We introduce an OR-precedence constraint.



We can interpret this as: j_F may start after one of the direct predecessors of has completed. Ordinary precedence constraints correspond to AND-precedence constraints.

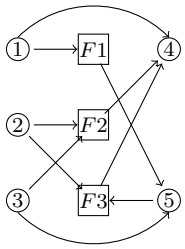


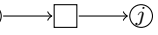
So j must wait for the completion of all direct predecessors.

Figure 1: Algorithm for Testing Feasibility

```
1: Intiate list L := [ ]
2: while there is a job i ∈ V that is not a waiting job of a condition in W do
3:     Insert i at the end of L and delete if from V
4:     if some waiting condition (X, j) becomes satisfied) then
5:         delete (X, j) from W
6:     end if
7: end while
8: return L
```

Example 17 :



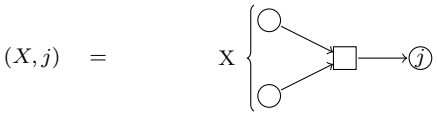
With the forbidden sets $F_1 = \{1, 5\}$, $F_2 = \{2, 3, 4\}$ and $F_3 = \{2, 4, 5\}$. We can also represent every precedence constraint $i < j$ as an OR precedent constraint. 

The above network is defined by the selection $s = (5, 4, 4)$. ♢

Consequences

(1) The resulting AND/OR network is bipartite.

(2) We may just speak of a systems W of waiting conditions.



So (X, j) is a system of waiting conditions if and only if it corresponds to a bipartite AND/OR network.

Definition 9.3:
*A **realization** of W is an acyclic graph $R = (V, A)$ on V such that for every waiting condition (X, j), there is some $i \in X$ preceding j. We call W **feasible** if and only if W has a realization (if and only if W has a linear realization).*

The following algorithm tries to construct a linear realization for W. It imitates topological sort for digraphs.

Lemma 9.4: Correctness of the algorithm 1
W is feasible if and only if L contains all jobs.

Proof.
$\Leftarrow$ trivial

$\Rightarrow$ Let R be a linear realization but $L \neq V$.

Let j be the first job in $V\backslash L$ (= red jobs). Since j is not in L there is a waiting condition (X, j) with $X \subseteq V\backslash L$. Thus all jobs in X are red. But in R there must be a job $i \in X$, i.e. a red job before j. This gives a contradiction $\lightning$

□

Corollary 9.5: Consequences
The set $V\backslash L$ corresponds to a generalized cycle C. Meaning, for $j \in C$ there exist waiting conditions (X, j) with $X \subseteq C$.

(1) The list L is an $\subset$-maximal feasible subset of V.

(2) Every linear realization can be constructed by the algorithm.

Proof.

(1) Let $V\backslash L \neq \emptyset$ set $C := V\backslash L$ and let W_C denote the set of all waiting conditions (X, j) with $X \subseteq C$ and $j \in C$. Then W_C is a relaxation of the original problem W. So no job of C can be in any linear realizer of W_C. Therefore no job of C can be in any linear realizer of W. Thus L is an $\subseteq$-maximal set of jobs that can be in a linear realizer.

(2) Choose jobs in the algorithm according to the linear realizer R. Then the algorithm produces R.

□

Remark:
The linear realizations form the basic words of an antimatroid ([KL84]). △

Question: Does every feasible selection define a policy?

Lemma 9.6:
Let s be a feasible selection for $[G, \mathcal{F}]$. Then, for every $x \in \mathbb{R}^n_>$, the earliest start ES_W with respect to the system W of waiting conditions given by s and G is well defined.

Proof. Since the selection s is feasible, also W is feasible. Consider the OR-nodes as dummy jobs. Let $(S_1, \ldots, S_N) = S$ be a vector of times associated with the given jobs $j \in V$ and the dummy jobs. A schedule S is feasible for W if and only if

$$S_w \geq \min_{j \in \mathrm{ImPred}(w)} (S_j + d_{jw}) \quad \text{for all OR-nodes } w$$
$$S_j \geq \max_{w \in \mathrm{ImPred}(j)} (S_w + d_{wj}) \quad \text{for all AND-nodes } j$$

This min-max system is fulfilled by S for $d_{jw} = x_j$ and $d_{wj} = 0$.
If W is feasible then every linear realizer R defines a solution of the min-max system. With lemma 9.7 there exists a unique minimal feasible solution $S = (S_1, \ldots, S_N) \geq 0$. Clearly every S_j for $j \in V$ is the earliest start of j with respect to W. □

Lemma 9.7:
Let $(S_1, \ldots, S_N)$ and $(T_1, \ldots, T_N)$ be two solutions of a general min-max system. Then

$$(\min\{S_1, T_1\}, \ldots, \min\{S_N, T_N\})$$

is also a solution.

Proof. Consider an OR-node w. Then there exist AND-nodes i and k with

$$S_w \geq S_i + d_{iw} \qquad T_w \geq T_k + d_{kw}$$

Let the minimum in

$$S_w\,,\, T_w \geq \min\{S_i + d_{iw}, T_k + d_{kw}\}$$

be without loss of generality be attained by $S_i + d_{iw}$. This implies

$$\min\{S_w, T_w\} \geq S_i + d_{iw} \geq \min\{S_i, T_i\} + d_{iw}$$

Therefore is the OR inequality for node w fulfilled.
Consider an AND-node j. Then there exist OR-nodes w with

$$S_j \geq S_w + d_{wj} \qquad T_j \geq T_w + d_{wj} \qquad \forall w$$

For all w and w.l.o.g. $S_j \leq T_j$ we get

$$\min\{S_j, T_j\} \geq S_j \geq \min\{S_w, T_w\} + d_{wj}$$

Therefore is the AND inequality for node j fulfilled. □

Theorem 9.8:
Every feasible selection s for $[G, \mathcal{F}]$ defines a preselective policy.

Proof. Consider a feasible selection s and W the associated waiting conditions for s, G. Let Π by the planning rule induced by s. Then we know by the lemmas above $\Pi = ES_W$. The proof of the (NA) property is similar to that for ES-policies (theorem 7.3). Let x, y look the same to Π at time t and $\Pi[X](j) = t$. Meaning $x \sim_t y$

$$\begin{aligned} x_i &= y_i \quad \text{for all jobs } i \text{ completed before } t \\ \bar{x}_i &= \bar{y}_i \quad \text{for all busy jobs} \end{aligned}$$

Thus all waiting conditions are fulfilled with the same times with respect to x and y which implies $\Pi[y](j) = t$. □

Question: What is the relationship with ES-policies?

Theorem 9.9:
Let s be a feasible selection for $[G, \mathcal{F}]$, W be the associated system of waiting conditions and Π be the associated policy. Then

$$\Pi = ES_W = \min\{ES_R \mid R \text{ is a realizer of } W\}$$

Proof. Consider $F \in \mathcal{F}$ and consider the corresponding OR-node w. Let $j \in F$ be the waiting job. Then there is some $i_0 \in \mathcal{F}\backslash\{j\}$ such that $S_j = S_{i_0} + x_{i_0}$. So if we delete all other arcs except (i_0, w) we get the same minimal solution $ES_W = S$ for the given x. Therefore S is a minimal solution to the modified problem which is tighter. Thus S is the unique minimal solution for the modified problem. Hence F is settled by letting j wait for i_0. This implies that $[G + (i_0, j), \mathcal{F}\backslash\{F\}]$ has the same minimal solution S for x.
Iteration gives that

$$[G + \{(i_F, j_F) \mid F \in \mathcal{F}\}, \emptyset]$$

has the same minimal solution S for x.

Claim:
$R := G + \{(i_F, j_F) \mid F \in \mathcal{F}\}$ is a realizer of W.

Proof. We have added one arc (i_F, j_F) for every waiting condition coming from $\mathcal{F}$. For precedence constraints $i \to \square \to j$ a choice (i, j) is forced in every realizer. So our choices lead to an acyclic graph since all $x_j > 0$. If not, the cycle would have positive length and some waiting condition is not satisfied. □

So $S_j = ES_R[x](j)$ and since R is acyclic and ES is the minimal solution. □

Question: How to compute ES_W? → Section 10
Question: Do preselective policies have nice properties?

Theorem 9.10:
Let Π be a policy for $[G, \mathcal{F}]$ then the following are equivalent

(1) Π *is preselective.*

(2) Π *is monotone, i.e.* $x \leq y \Rightarrow \Pi[x] \leq \Pi[y]$.

(3) Π *is continuous.*

Proof. Here we may only prove (1) ⇒ (2) and (1) ⇒ (3) and the rest in section **??**. Since ES_R is continuous, monotone and convex, the claim follows with

$$\Pi = \min\{ES_R \mid R \text{ realizer of } W\}$$

because monotonicity and continuity is preserved by taking the minimum. □

Remark:
The points (2) and (3) show that Graham anomalies of type a) occur in pairs. △

Exercises

(9.1) Theorem 9.9 shows that a preselective policy is the minimum of ES-policies. Consider now a set $\mathcal{H}$ of feasible orders and set $\Pi = \min\{ES_H \mid H \in \mathcal{H}\}$. Give necessary and sufficient conditions (on $\mathcal{H}$) for Π being a policy.

(9.2) Let $OPT^{\text{Pres}}(\kappa, Q) = \min\{E_Q(\kappa^\Pi) \mid \Pi \text{ preselective}\}$ be the optimum value over the class of preselective policies. Show that there are instances with

$$OPT^{\text{Pres}}(\kappa, Q) < OPT^{\text{ES}}(\kappa, Q)$$

but that OPT^{Pres} and OPT^{Prior} are incomparable in general.

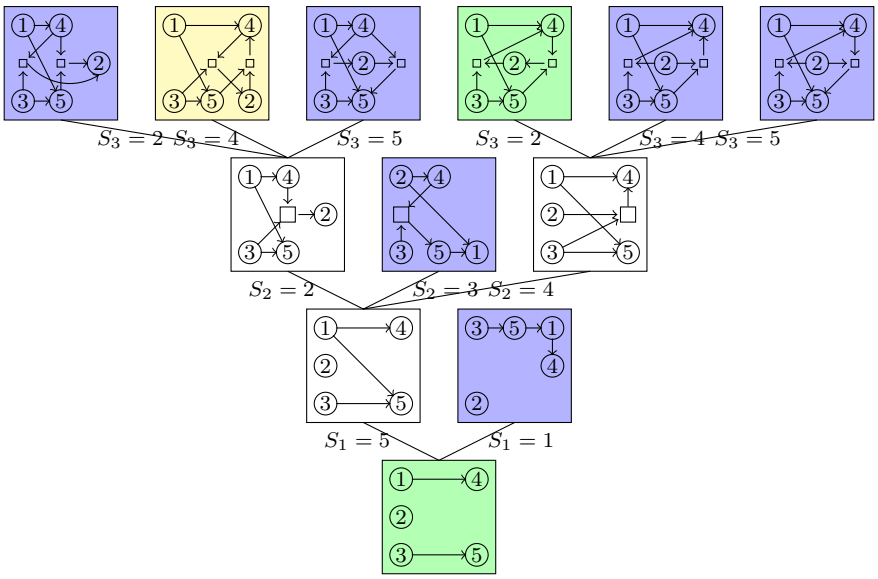
10 Constructing and Evaluating Preselective Policies

The goal of this section is a systematic construction similar to ES-policies along a conflict settling tree. Our tree consists of

- A **root** G.
- Some **nodes** that correspond to AND-OR networks arising from choices of waiting jobs on some forbidden sets.
- The **children** of a node D are all AND-OR networks obtained from D by choosing awaiting job from one yet unsettled forbidden set. We need to check the unsettledness and may use a suitable ordering of the forbidden sets.

Let us recall a previous example.

Example 18 :
Conflict settling tree



We use the ordering $F_1 = \{1, 5\}$, $F_2 = \{2, 3, 4\}$ and $F_3 = \{2, 4, 5\}$. We can handle F_1 as a simple precedence constraint since $|F_1| = 2$. At different depths of the tree we get blue acyclic feasible networks and yellow feasible networks starting from the green root. ♢

The task to check if a forbidden set F is already settled by the partial selection $(S_1, \ldots, S_r)$ corresponds to finding forced waiting condition of the form $(F\backslash\{i\}, j)$.

Definition 10.1:
The job j is forced to wait for U if and only if all linear realizers have some $u \in U$ before j.

Figure 2: An algorithm for finding forced waiting conditions

Input: Jobs V, feasible waiting conditions W and a set $U \subseteq V$
Output: A list L of jobs

1: list $L := [] = \emptyset$
2: **while** there is a job $i \in V \backslash U$ that is not a waiting job in W **do**
3: Insert i at the end of L and delete i from $V \backslash U$
4: **if** some waiting condition (X, j) becomes satisfied **then**
5: delete (X, j) from W (delete all such waiting jobs)
6: **end if**
7: **end while**
8: **return** L

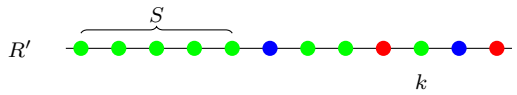
Correctness of the Algorithm

Theorem 10.2:
(U, j) is a forced waiting condition if and only if $j \notin L$ (and $\notin U$).

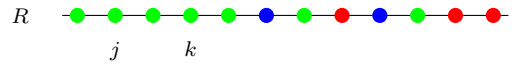
Lemma 10.3:
There is a linear realization of W starting with all jobs j for which (U, j) is not a forced waiting condition.

Then only U and forced waiting jobs are not in L at termination of the algorithm.

Proof. For now let us call jobs that are in a forced waiting condition with U red and those that are not in a forced waiting condition with U green. We call jobs in U blue. Now consider the linear realization R' constructed by our algorithm



With the maximal initial segment S of green jobs. Then the first job after S must be blue. Let k be a green job after S then there is a linear realization R with U after k.



Then there must be a first green job j that is in R before k but was not in S. Thus there is a waiting condition (X, j) with X after S in R'. Therefore there is some $i \in X$ before j in R. But all jobs before j in R are in S. This gives a contradiction $\lightning$. □

Therefore we can detect forced waiting conditions in linear time.

Computing earliest start times for a preselective policy

Input: preselective policiy Π, processing time vector x
Output: vector $\Pi[x]$ of starting times. This corresponds to an earliest start with respect to the system of waiting conditions given by selection S defining Π and a graph G of precedence constraints. We translate this to an algorithm on the

Figure 3: Algorithm 3

1: Set $S_j := 0$ if there is no $(w, j) \in A$
2: For OR nodes $w \in \text{out}(j)$ set $S_w = \min\{S_j + d_{jw} \mid (j, w) \in A\}$ and set $S_w = \infty$ otherwise.
3: **loop**
4: Choose unmarked OR node $w = (X, j)$ with minimum S_w and mark w
5: Reduce indegree of j by 1
6: **if** indegree $(j) = 0$ **then**
7: Set $S_j := \max\{S_w \mid (w, j) \in A\}$
8: For unmarked OR nodes $w \in \text{out}(j)$ set $S_w = \min\{S_j + d_{jw} \mid (j, w) \in A\}$
9: **end if**
10: **end loop**

AND/OR network representing Π and G. The previous section 9 allows us to solve a system of min-max inequalities and compute the unique componentwise minimal solution.

- General Case: arbitrary arc weights d_{jw}, d_{wj} (also negative)
- Special Case: (needed here) We have positive arc weights $d_{jw} := x_j$ for AND node j and OR node w. We may assume the arc weight $d_{wj} = 0$ for w OR node and j AND node. Since there is only one outgoing arc from every OR node.

So we can solve the special case using a Dijkstra like algorithm.

Theorem 10.4:
The algorithm 3 computes the unique minimal feasible solution ≥ 0 of the min-max system given by the AND /OR graph $D = (V \cup W, A)$ in

$$O(|V| + |W| \cdot \log |W| + |A|)$$

Proof. We leave the run time as an exercise and will only prove *correctness*.
Let S be the vector of start times constructed by the algorithm. Let S^* be the ES-vector (see lemma 9.6) of (V, W). Assume $S \neq S^*$. Then chose a node v with $S_v > S_v^*$ and S_v^* minimum.

Case 1 v is an AND node
Then there exists an OR node $w = (X, v)$ with

$$S_w = S_v + \underbrace{d_{wv}}_{0} > S_v^* \geq S_w^*$$

By the choice of v we get $S_v^* = S_w^*$ and $S_w > S_w^*$. Thus we reduced the problem to the case that v is an OR node (case 2).

Case 2 v is an OR node
There exists an AND node i with

$$S_v^* = S_i^* + d_{iv} \qquad d_{iv} > 0$$

Claim:
$S_i > S_i^*$

Suppose not, i.e. $S_i = S_i^*$. Then, when the algorithm asigns to i the value S_i, S_v is set to

$$\min_{(j,v)}\{S_j + d_{jv}\} \leq S_i^* + d_{iv} = S_v^*$$

if unmarked and if already marked we have

$$S_v \leq S_i + d_{iv} = S_i^* + d_{iv} = S_v^*$$

Either way we get a contradiction to $S_v > S_v^*$.
So with the claim we know $S_i > S_i^*$ and $S_i^* < S_v^*$ which contradicts the choice of v ↯.

□

For a min-max system $(\infty, \ldots, \infty)$ is a solution.

Definition 10.5:
*We call a solution S of a min-max system **feasible** if every $S_j < \infty$ and the min-max system **feasible** if there is a feasible solution.*

Lemma 10.6:
A min-max system with $x_{jw} > 0$ and $x_{wj} \geq 0$ has a feasible solution S if and only if the waiting conditions are feasible (feasibility of min-max system = structural feasibility).

Proof.

⇒ All $x_{jw} > 0$ implies that for every $w = (X, j)$ there is an $i \in X$ with $S_i < S_w \leq S_j$. (Otherwise S is not feasible.) The arcs (i, j) define a realization of W. If not, they contain a cycle and we would have $S_l < S_k$ for every arc (l, k) on the cycle, which cannot be the case.

⇐ W is feasible implies that there exists a realization R. Now ES_R defines a feasible solution of the min-max system.

□

We consider a min-max system

$$\left.\begin{array}{llll} j \in V & \text{AND node} & S_j \geq \max\{S_w + d_{wj} \,|\, (w,j) \in A\} \\ w \in W & \text{OR node} & S_w \geq \min\{S_j + d_{jw} \,|\, (j,w) \in A\} \end{array}\right\} \text{ min-max system}$$

We allow $(\infty, \ldots, \infty)$ as a solution. If $(S_1, \ldots, S_n)$ and $(T_1, \ldots, T_n)$ are solutions then $(\min\{S_1, T_1\}, \ldots, \min\{S_n, T_n\})$ is a solution as well.

Remark:
There is a unique componentwise minimal solution $S \geq 0$ and there is a unique maximal feasible subset of jobs. △

Claim:
Any feasible schedule $S = (S_1, \ldots, S_n)$ is a certificate for SOLVABILITY $\in NP$.

Sketch of Proof. Let $S = (S_1, \ldots, S_n)$ be the unique minimal solution (maybe with ∞). For every AND node j one can delete all but one incoming arcs without changing S_j. Then every cycle has non-negative length.
Relaxing in every AND node gives

(1) A relaxed problem with only min inequalities (OR nodes)

(2) We can check $S_j > K$ by shortest path algorithms in polynomial time.

(3) The relaxed is a certificate for SOLVABILITY $\in$ coNP.

Relaxing
Delete all possible arcs such that S^* does not change.
Suppose there exists an AND node j with indegree > 1 let S^k be the best schedule after deleting (w_k, j). Assume without loss of generality $k = 2$ and $S_j^1 \leq S_j^2$. We also have by construction $S_j^* > S_j^k$. Now define

$$S_i := \min\{S_i^1 + S_i^2 - S_j^1, S_i^2\} \qquad \text{for all nodes}$$

This gives $S \leq S_2$ and $S_j = S_j^2$. Now we want to show that S is a schedule.

$$\left.\begin{array}{rll} w \neq w_2 & (w,j) \Rightarrow & S_j = S_j^2 \geq S_w \\ & (w_2,j) \Rightarrow & S_j = S_j^1 + S_j^2 - S_j^1 \geq S_{w_2} \end{array}\right\} \Rightarrow S_j \geq \max_{(w,j)} S_w$$

Now for AND node $\neq j$ we have

$$S_k^1 \geq S_w^1 \text{ and } S_k^2 \geq S_w^2 \;\Rightarrow S_k^1 + S_k^2 - S_j^1 \geq S_w^1 + S_w^2 - S_j^1$$

A similar arguement for OR nodes gives that S is a schedule. But then $S_j = S_j^2 < S_j^*$ contradicts the fact that S_j^* is the best schedule by construction. □

Theorem 10.7:
A schedule and a tightened subproblem are polynomially checkable certificates for membership in NP and coNP. Therefore SOLVABILITY $\in NP \cap coNP$.

Remark:
No polynomial algorithm known. It is not known to be NP-complete or coNP-complete. △

Exercises

10.1 Show that algorithm 3 can be implemented to run in

$$O(|V| + |W| \cdot \log|W| + |A|)$$

10.2 Derive a polynomial-time algorithm for finding the unique minimal (feasible) solution ≥ 0 of a min-max system with non-negative arc weights.
Hint: Try to relate the feasibility of the min-max system to the structural feasibility in the sense of lemma 10.6. What changes?

10.3 Derive a pseudo-polynomial-time algorithm for finding the unique minimal (feasible) solution ≥ 0 of a min-max system with arbitrary arc weights.

11 Characterization of ES- and Preselective Policies

So far we have shown that ES-policies are convex, continuous and monotone and that preselective policies are continuous and monotone. Now we show that they are already characterized by these properties.

Theorem 11.1:
Every monotone policy is dominated by a preselective policy.

Proof. Consider $[G, \mathcal{F}]$. Let A be an antichain of G, and x be a duration vector. We call a job j selected for (A, x) if and only if j waits for any $i \in A$ for all y that are only larger on A ($y \geq_A x$), i.e. for every such y there is $i \in A$ (may depend on y) with

$$\Pi[y](i) + y_i \leq \Pi[y](j)$$

Intuitively this notion means: For x several jobs might wait and making jobs of A longer reveals the *real* selected job.
Let $S(A, x)$ be the set of jobs selected for (A, x). First we want to prove

$$F \in \mathcal{F} \Rightarrow S(F, x) \neq \emptyset \qquad \forall \text{ policy } \Pi \tag{11.1}$$

Consider x^m (long on F) with

$$x_k^m := \begin{cases} x_k & k \notin F \\ x_k + m & k \in F \end{cases}$$

Since Π is a policy there is a job j_m that waits with respect to x^m. Thus some job j occurs infinitely often in the sequence $(j_m)_m$.

Claim:
$j \in S(F, x)$

Proof of the claim. Suppose not, say j does not wait with respect to y with $y \geq_F x$. Then

$$\Pi[y](j) < \min_{\substack{i \in F \\ i \neq j}} \{\Pi[y](i) + y_i\}$$

This means that the start of job j under Π remains the same if we enlarge the processing times on F, because we have the same history up to the start of j. Therefore j does not wait for all x^m with $x^m \geq_F x$. This is a contradiction □

We proved furthermore

$$\text{non-waiting is invariant under } F\text{-monotonicity} \tag{11.2}$$

Next we want to prove

$$(\forall F, x, y : x \leq y \Rightarrow S(F, x) \subset S(F, y)) \Rightarrow \Pi \text{ preselective} \tag{11.3}$$

Suppose the assumptions hold but Π is not preselective. Then there is an $F \in \mathcal{F}$ such that Π is not preselective on F. Then for every $j \in F$ there is x^j such that

j does not wait with respect to x^j. Now (11.2) gives $j \notin S(F, x^j)$ for all $j \in F$. Consider x defined component wise by

$$x_k := \min_j x_k^j$$

Let $j_0 \in S(F, x) \neq \emptyset$. Then $x \leq x^{j_0}$ and the assumptions implies $j \in S(F, x^{j_0})$ by a diagonalization arguement. But this is a contradiction. Thus (11.3) holds.
Further we want to prove

$$\Pi \text{ monotone} \Rightarrow (\forall F, x, y \,:\, x \leq y \Rightarrow S(F, x) \subset S(F, y)) \tag{11.4}$$

Suppose the conclusion does not hold. Then there is F, x, y with $x \leq y$ but $S(F, x) \nsubseteq S(F, y)$. Thus there exists $j \in S(F, x) \backslash S(F, y)$. Consider x^m, y^m as above ($+m$ on jobs in F). So j waits for x and thus for all x^m. But j does not wait for some $y' \geq_F y$ and thus j does not wait for all $y^m \geq_F y'$ by (11.2).
This implies the contradiction

$$m \leq \Pi[x^m](j) \leq \Pi[y^m](j) = \Pi[y'](j)$$

□

Corollary 11.2:
Let Π *be an arbitrary policy then* Π *is monotone if and only if* Π *is preselective up to dominance (i.e. maybe not earliest start).*

Theorem 11.3:
Every continuous and elementary policy is preselective.

Corollary 11.4:
Let Π *be an arbitrary policy. Then* Π *is preselective if and only if* Π *is monotone (up to dominance) if and only if* Π *is continuous and elementary.*

Consequences

(1) Graham anomalies of type a) come in pairs.

(2) Preselective policies form a natural class fulfilling stability.

Theorem 11.5:
If Π *is convex implies then it is an ES-policy.*

Proof.

(1) Π convex $\Rightarrow$ Π is dominated by a preselective policy.

Claim:
For all x, y, F we have $x \leq y$ implies $S(F, x) \subset S(F, y)$.

Let $x \leq y$ but $j \in S(F,x)\backslash S(F,y)$. Consider the line $x(\lambda) = (1-\lambda)x + \lambda y$ for $0 \leq \lambda \leq 1$. Let λ^* be the first λ with $j \notin S(F, x(\lambda))$. Without loss of generality we assume x, y are close enough on the line to $x(\lambda^*)$ such that $z := 2x - y > 0$ meaning

$$x = \frac{1}{2}y + \frac{1}{2}z$$

Consider x^m, y^m as before (add m to jobs in F). Then

$$x^m = \frac{1}{2}y^{2m} + \frac{1}{2}z = \frac{1}{2}y^{2m} + x - \frac{1}{2} = \frac{1}{2}y + \mathbb{1}_F^m + x - \frac{1}{2}y$$

holds and gives

$$m \leq \Pi[x^m](j) \leq \frac{1}{2}\Pi[y^{2m}](j) + \frac{1}{2}\Pi[z](j) \overset{(11.2)}{=} \frac{1}{2}\Pi[y](j) + \frac{1}{2}\Pi[z](j)$$

a contradiction.

(2) Π convex $\Rightarrow \exists$ ES-policy $\Pi^* \leq \Pi$.
By (1) the convexity implies that Π is preselective. Thus there is a waiting job j on $F \in \mathcal{F}$.

Claim:
j waits always for the same job i.

Suppose not. Then for every $i \in F\backslash\{j\}$ there is x^i with

$$\Pi[x^i](i) + x^i_i > \Pi[x^i](j)$$

meaning job j does not wait for i with respect to x^i. Consider $x^{i,m}$ (again make jobs in F larger by m). Then by monotonicity of non-waiting $\Pi[x^{i,m}](j)$ does not change. Now define

$$z^m := \frac{1}{|F|-1}\sum_{i \in F\backslash\{j\}} x^{i,m}$$

(convex combination). This leads to

$$m \leq \Pi[z^m](j) \leq \frac{1}{|F|-1}\sum_{i \in F\backslash\{j\}} \Pi[x^{i,m}](j)$$

Therefore the start is larger than or equal to the first completion on $F\backslash\{j\}$. Every i is at least $\frac{m}{|F|-1}$ long with respect to $x^{i,m}$ and hence at least m long with respect to z. But this is a contradiction.
So every F is settled by a waiting pair $i < j$ and there is an ES-policy $\Pi^* \leq \Pi$.

(3) Π is elementary.
Suppose not. Then there exist x^0 and j_0 such that j_0 starts in $\Pi[x^0]$ at a time t where no job $j \neq j_0$ ends. We define in $\Pi[x^0]$ the sets $C(t)$ of completed jobs before t and $B(t)$ of busy jobs at t. Further we define the set

$$X := \left\{ x \in \mathbb{R}^n_> \,\middle|\, x_j = x^0_j \quad \forall j \in C(t) \text{ and } x_j > (t - \Pi[x^0](j)) \quad \forall j \in B(t) \right\}$$

of all $x \in \mathbb{R}^n_>$ that look the same to Π at t. So we have

$$\Pi[x](j_0) = t \quad \forall x \in X$$

Now let $z \in \mathbb{R}^n_>$ with $z_j = x^0_j$ for all $j \in (C(t) \cup \mathrm{Suc}_G(j_0) \cup \{j_0\})$ and z_j short for all other jobs. Hereby short means that as these jobs end before t, say at $t - \varepsilon$. So only job j_0 and its successors are left. $\Pi[z]$ has the same history as $\Pi[x^0]$ and $z \notin X$.
Since policies have no total idle time and j is the only job that can be started we can conclude $\Pi[z](j_0) = t - \varepsilon < t$. If we now choose $y \in X$ with $\frac{1}{2}y + \frac{1}{2}z \in X$ (choose y_j large for $j \in B(t)$ in $\Pi[x^0]$) we get

$$t = \Pi[\frac{1}{2}y + \frac{1}{2}z](j_0) \leq \frac{1}{2}\Pi[y](j_0) + \frac{1}{2}\Pi[z](j_0) < t$$

a contradiction.

(4) Π is an ES-policy.
Consider the most restrictive problem $[G^*, \mathcal{F}^*]$ for which Π is still a policy.

- Add to G all $i < j$ with $\Pi[x](i) + x_i < \Pi[x](j)$ for all x
- Let $\mathcal{F}^*$ be all antichains of G^* that are not scheduled simultaneously by Π for all x.

Then Π is a convex policy for $[G^*, \mathcal{F}^*]$. By (1) there is an ES policy Π^* with $\Pi^* \leq \Pi$.

Claim:
$\Pi^* = \Pi$.

We prove this by induction along decision times t of Π for arbitrary fixed vector x.
For $t = 0$ we have $S^*(0)$ the set of jobs started by Π^* at $t = 0$ and $S(0)$ for Π. If $S^*(0) \neq S(0)$ implies with $\Pi^* \leq \Pi$ that there exists a $j \in S^*(0) \backslash S(0)$. Since Π is elementary j waits for another completion. Thus $S^*(0)$ is not processed simultaneously in $\Pi[x]$ and therefore $S^*(0)$ is either forbidden or not an antichain because $[G^*, \mathcal{F}^*]$ is most restrictive. But then Π^* is not a policy for $[G^*, \mathcal{F}^*]$ $\lightning$
Now the inductive step: We assume $S(t') = S^*(t')$ for all decision times $t' < t$. By the same argument as at $t = 0$ we can show $S(t) = S^*(t)$ for the next completion time t (decision time) (the same for Π and Π^*).

□

12 Set Policies

Set policies try to generalize priority and preselective policies while using only robust information. They are reasonably stable.

Definition 12.1:
Π *is a* ***set policy*** *for* $[G, \mathcal{F}]$ *if it is elementary and decisions at time* t *are only based on*

- *the set* $C(t)$ *of jobs completed before* t
- *the set* $B(t)$ *of jobs busy at* t

(so only the sets, not the processing times etc.)

Remark:
Static priority policies, ES-policies and preselective policies are set policies but not the only ones. △

Theorem 12.2: Representation Theorem
Let Π *be a set policy for* $[G, \mathcal{F}]$. *Then there exists a partition of* $\mathbb{R}^n_>$ *into finitely many sets* $Z_1, \ldots, Z_m$ *and finitely many interval orders* $G_1, \ldots, G_m$ *on* v *such that*

(1) Every Z_i *is a polyhedral (convex) cone.*

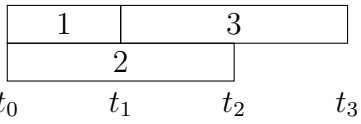
(2) $\Pi[x] = ES_{G_i}[x]$ *for all* $x \in Z_i$

i.e. set policies behave locally as ES-policies.

The following example will lead us through the proof.

Example 19 :
We consider $G = (V = \{1, 2, 3\}, \emptyset)$ and $\mathcal{F} = \{\{1, 2, 3\}\}$ and Π a priority policy for the list $L : 1 < 2 < 3$. Consider the vector $x = (1, 2, 2)$ of processing times. This gives

$\Pi[x] =$



and the sequence of sets

$$S = [(\emptyset, \emptyset), (\{1\}, \{2\}), (\{1, 2\}, \{3\}), (\{1, 2, 3\}, \emptyset)]$$

Now we define

$$Z_S = \left\{ x \in \mathbb{R}^3_> \,|\, x_1 < x_2, x_2 < x_1 + x_3 \right\}$$

♢

Proof. For given $[G, \mathcal{F}]$ and a fixed vector x of processing times, Π makes a sequence of decisions for x that depend only on sets. Let S be the sequence of these sets

$$S := [(C(t_0) = \emptyset, B(t_0) = \emptyset), \ldots, (C(t_k) = V, B(t_k) = \emptyset)]$$

Now given such a sequence S we define

$$Z_S := \{x \in \mathbb{R}^n_> \mid x \text{ induces the sequence } S\}$$

Claim:
The sets Z_S for a (finite) partition of $\mathbb{R}^n_>$.

This is clear since every x leads to exactly one S and there are only finitely many such sequences.
Now we want to show (1) and (2) for these sets.

(1) We proced inductively along the sequence. Every pair (C^i, S^i) represents the state of a decision at a time t_i that depends on x. But the state is the same for all $x \in Z_S$.

Claim:
The decision time t_i is equal to the sum of some x_r for $r \in C(t_i)$ and every such x_r occurs only once, meaning

$$t_i = \sum_{r \in C(t_i)} \alpha_r x_r \qquad \text{with } \alpha_r \in \{0, 1\}$$

and the jobs with $\alpha_r = 1$ are sequential in $\Pi[x]$.

Proof. By induction. We have $t_0 = 0$ and all $\alpha_r = 0$. Now at t_1 we get $t_1 = x_i$ for $i \in C(t_1)$.
Now look at t_i. Some jobs must complete at t_i since Π is elementary, say job j. Then the start of j is a decision time $t \leq t_{i-1}$ and thus $t_i = t_{\text{start } j} + x_j$ because $t_{\text{start } j}$ only involves other x_r that are sequential by induction hypothesis. □

Now we want to use this claim to express conditions on $x \in Z_S$ by homogeneous inequalities and equations. First we obviously have

$$x_j > x_i \quad \text{for all } j \in B(t_1)\,,\, i \in C(t_1)$$

and analogously for later times

$$j \in C(t_i) \backslash C(t_{i-1}) \Rightarrow t_i = x_j + t_{\text{start of } j}$$
$$k \in B(t_i) \Rightarrow t_i < x_k + t_{\text{start of } k}$$

Since Π is a set policy, these equations and inequalities induce exactly the sequence S. Therefore Z_S is the intersection of finitely many open halfspaces and linear spaces, all containing 0 in its closure. Thus Z_s is a polyhedral cone.

(2) S defines an abstract interval order G_S

$$i \;\to\; [t_{\text{start } i}, t_{\text{finish } i}]$$

These are both abstract points given by the sum of x_r of sequential jobs. The order of theses is the same for all $x \in Z_S$. So

$$\Pi[x] = ES_{G_S}[x] \qquad \text{for } x \in Z_s$$

□

Corollary 12.3:

(1) Graham anomalies occur only across the boundaries of the cones.

(2) Every Set policy is almost everywhere continuous.

(3) The class of set policies is stable for continuous probability distributions.

Definition 12.4:
*A cost function κ is **additive** if there is a set function $g : 2^V \to \mathbb{R}$ (the cost rate) with*

$$\kappa(C_1, \ldots, C_n) = \int g(U(t))\, dt$$

where $U(t)$ is the set of uncompleted jobs at t

Theorem 12.5:
If all jobs are exponentially distributed and independent and the cost function κ is additive, then there is an optimal set policy (optimal among all policies).[MRW85]

Sketch of Proof. We assume additionally that a prescribed set V' of jobs must start at time 0. Let Π be a policy for this version.

Claim:
There is a set policy Π^* for this version with

$$\mathbb{E}\left[\kappa^{\Pi^*}\right] \leq \mathbb{E}\left[\kappa^{\Pi}\right]$$

Proof of the Claim. By induction on $n = |V|$.
$\boldsymbol{n = 1}$ **:** There is only one policy: start 1 at 0. This is a set policy.
Inductive Step:
Consider the problem with n jobs and the start set V'. Assume that there is an optimal set policy for all problems with less than n jobs and arbitrary start set V''. There is a policy Π' with $\mathbb{E}[\kappa^{\Pi'}] \leq \mathbb{E}[\kappa^{\Pi}]$ that always waits for the first completion of a job since there are no tentative decision times at $t = 0$. (Part 1)
Suppose Π fixes a tentative decision time $t < \infty$ at time 0. Consider

$$X = \{x \in \mathbb{R}^n \,|\, \text{every job started at 0 is longer than } t\}$$

Then Π starts some new jobs at t and thus

$$\mathbb{E}\left[\kappa^{\Pi}\right] = Q(X) \cdot \mathbb{E}\left[\kappa^{\Pi} \,\middle|\, X\right] + Q(X^C) \cdot \mathbb{E}\left[\kappa^{\Pi} \,\middle|\, X^C\right]$$

The memoryless property of the exponential distribution further gives

$$\mathbb{E}\left[\kappa^{\Pi}\right] = t \cdot g(V) + \mathbb{E}\left[\kappa^{\Pi}\right]$$

Consider Π' that starts all jobs at 0 and otherwise behaves as Π.

$$\Rightarrow \mathbb{E}\left[\kappa^{\Pi}\right] - \mathbb{E}\left[\kappa^{\Pi'}\right] = Q(x) \cdot t \cdot g(V) > 0$$

This proves (Part 1). So without loss of generality let Π wait for the first completion. Consider a particular set C of jobs to complete first. Let Π^C be the policy for the subproblem on $V \backslash C$ with start set V'' induced by Π. Now we get

$$\begin{aligned}
&\mathbb{E}\left[\kappa^{\Pi}\right] \\
&\quad = \sum_{\text{all possible } C} Q(\text{jobs in } C \text{ end first}) \cdot (g(V) \cdot (\text{expected completion of } C)) \\
&\qquad = \sum_{j \in V} Q(j \text{ ends first}) \cdot \left(\mathbb{E}[j \text{ completes}] \cdot g(V) + \mathbb{E}[\kappa^{\Pi^j}]\right) \\
&\qquad = \sum_{j \in V} \frac{\lambda_j}{\lambda_1 + \ldots + \lambda_n} \cdot \left(\frac{1}{\lambda_j} \cdot g(V) + \mathbb{E}[\kappa^{\Pi^j}]\right) \qquad (12.1)
\end{aligned}$$

We use the inductive hypothesis on $\mathbb{E}[\kappa^{\Pi^j}]$, so we can replace Π^j by an optimal set policy for the subproblem. Combining this for all subproblems gives a set policy Π^* for given problem with $\mathbb{E}[\kappa^{\Pi^*}] \leq \mathbb{E}[\kappa^{\Pi}]$. □

Since there are only finitely many set policies, there is an optimal set policy and with $V' = \emptyset$ also for the original problem. □

Theorem 12.6:

(1) $\kappa = C_{\max}$ *is additive with*

$$g(U) := \begin{cases} 1 & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset \end{cases}$$

and LEPT is optimal for $P|p_j \sim \exp|C_{\max}$.

(2) $\kappa = \sum_j w_j C_j$ *is additive with*

$$g(U) \sum_{j \in U} w_j$$

and SEPT is optimal for $P|p_j \sim \exp|\sum C_j$

Example 20 :
We assume no precedence constraints, $m = 3$ identical machines, 6 jobs $X_j \sim \exp(a_j)$ with $a_1 = a_2 = a > 1$ and $a_3 = a_4 = a_5 = a_6 = 1$. We consider the set function

$$g(U) := \begin{cases} w \gg 1 & \text{if } 1 \in U \text{ and } 2 \in U \\ 1 & \text{if } 1 \notin U \text{ and } (2,5 \in U \text{ or } 2,6 \in U) \\ 1 & \text{if } 2 \notin U \text{ and } (1,3 \in U \text{ or } 1,4 \in U) \\ 0 & \text{otherwise} \end{cases}$$

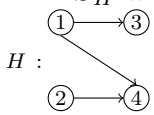
The optimal set policies involves deliberate idleness. ♢

Non-idleness problem Under which condition on $g : 2^V \to \mathbb{R}$ does there exist an optimal set policy without idleness?

Remark:
The formula (12.1) in the proof of theorem 12.5 gives an algorithm for computing the expected cost of a set policy for an additive cost function κ. △

Exercises

(12.1) Calculate the expected makespan by the algorithm in the remark above for the ES-policy $\Pi = ES_H$ with



Consider $X_j \sim \exp(\lambda_j)$ for
$\lambda_1 = 1$, $\lambda_2 = 2$, $\lambda_3 = 2$ and $\lambda_4 = 1$.

13 Simple Policies for the Expected Makespan

Determining an optimal policy is in general NP-hard. Therefore we want approximation algorithms. For an instance I of a scheduling problem an algorithm should produce a feasible solution $A(I)$ in form of a policy Π. Its value is given by the expected cost of Π. We want to give performance guarantees of the form

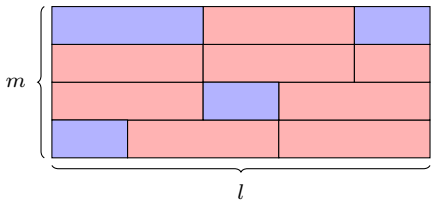
$$A(I) \leq \varrho \cdot OPT(I)$$

for machine scheduling problems.

Theorem 13.1:
Let G be arbitrary and the set $\mathcal{F}$ of forbidden sets corresponds to m-machines. Further let the cost function be the makespan, i.e. $\kappa = C_{\max}$. Then for every static priority rule Π and every vector x of processing times the following asymptotically tight bound holds:

$$C^{\Pi}_{\max}(x) \leq \left(1 + \frac{m-1}{m}\right) OPT(x) \; .$$

Proof. Consider $\Pi[x]$ as an $m \times l$ rectangle, where l denotes the makespan.



Let I_k be the blue idle time and B_k be the red busy time of machine k.

$$m \cdot l = \sum_{k=1}^{m} (|I_k| + |B_k|)$$

Case 1 No idle time implies that $\Pi[x]$ is optimal.

Case 2 Consider job j_0 that ends last.

Case (a) All idle times are parallel to j_0. Thus all machines are busy until the start of job j_0.

$$\begin{aligned} \Rightarrow \sum_{k=1}^{m} |I_k| &\leq (m-1)x_{j_0} && \leq (m-1)OPT \\ \Rightarrow \quad l \cdot m = \sum_{k=1}^{m} |I_k| + \sum_{k=1}^{m} |B_k| &\leq (m-1)OPT + mOPT \\ \Rightarrow \qquad l &\leq (1 + \frac{m+1}{m})OPT \end{aligned}$$

Case (b) There are idle times before the start of j_0. Consider the last such idle time with respect to its start. Why was j_0 not started in that idle time? It can only be, because a predecessor j_1 of j_0 was busy until the

start of j_0 or during an end interval of the idle time. So inductively we get a chain C with length l_C:

$$j_r <_G \ldots <_G j_1 <_G j_0$$

such that all idle times occur in parallel to the processing of that chain.

$$\sum_{k=1}^{m} |I_k| \leq (m-1) \cdot l_C \leq (m-1)OPT$$

Proceed as above.

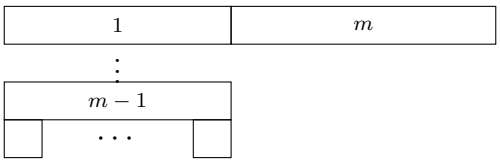
□

Remark:
The bound $(1 + \frac{m-1}{m})$ is achieved pointwise. Thus $\mathbb{E}(C^{\Pi}_{\max})$ is less or equal to the optimum expected over all policies for every joint distribution of processing times. △

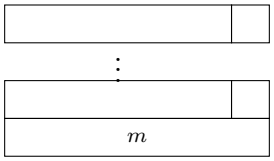
Example 21 :
To prove the tightness of the bound for m machines and no precedence constraints consider

jobs	$1, \ldots, m-1$	m	$m+1, \ldots, 2m-1$
processing times	$m-1$	m	1

For the priority list $L = 1 < \ldots < m-1 < m+1 < \ldots < 2m-1 < m$ the algorithm gives a policy $\Pi[x]$ with $C_{\max} = 2m + 1$.



Where as the optimal solution $OPT(x)$ only takes $C_{\max} = m$.



This gives the needed ratio of $1 + \frac{m-1}{m}$. ♢

14 Simple Policies for Weighted Completion Times

Here we need more sophisticated methods. We will use LP-guided construction of a policy. Let us first consider deterministic times for fixed vector $x = (p_1, \ldots, p_n)$ (fixed processing times).

Model:

We consider no precedence constraints and m machines. This can be generalized to include release dates $r_j \geq 0$. Consider the following LP in completion time variables C_j^{LP}.

$$\min \sum_{j=1}^{n} w_j C_j^{LP}$$

$$\text{s.t.} \quad \sum_{j \in A} p_j \cdot C_j^{LP} \geq \frac{1}{2m} \cdot \left(\sum_{j \in A} p_j \right)^2 + \frac{1}{2} \sum_{j \in A} p_j^2 \quad \forall A \subseteq V \tag{14.1}$$

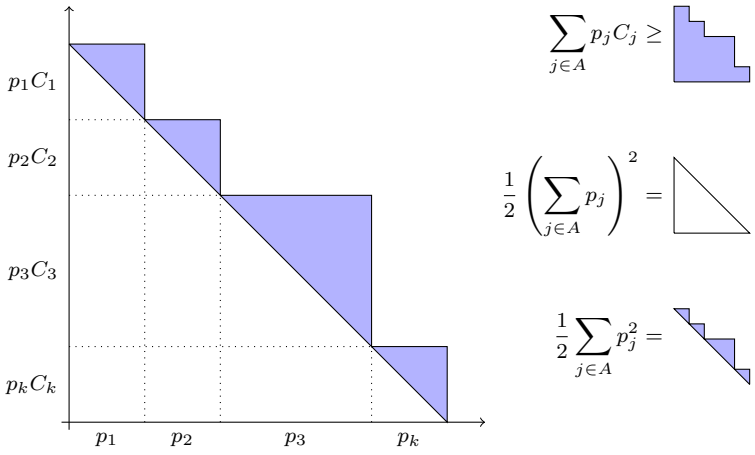
$$C_j^{LP} \geq p_j \qquad \forall j \in \{1, \ldots, n\} \tag{14.2}$$

Lemma 14.1:

The completion times of every feasible schedule fulfill the LP constraints (14.1) *and* (14.2).

Proof. The second constraint is always fulfilled. Now consider a feasible schedule S with completion times $C_1, \ldots, C_n$.

$m = 1$ The left hand side is smallest if the jobs of A are in the beginning. So consider the picture $A = \{1, \ldots, k\}$:



So with $m = 1$ obviously holds

$$\sum_{j \in A} p_j \cdot C_j^{LP} \geq \frac{1}{2 \cdot 1} \cdot \left(\sum_{j \in A} p_j \right)^2 + \frac{1}{2} \sum_{j \in A} p_j^2$$

$m > 1$ We reduce the problem to the 1 machine case and use the Cauchy-Schwarz inequality.

$$\begin{aligned}
\sum_{j \in A} p_j C_j &= \sum_{i=1}^{m} \sum_{j \in A \cap M_i} p_j C_j \\
&\geq \sum_{i=1}^{m} \left(\frac{1}{2} \left(\sum_{j \in A \cap M_i} p_j \right)^2 + \frac{1}{2} \left(\sum_{j \in A \cap M_i} p_j^2 \right) \right) \\
&= \frac{1}{2} \sum_{i=1}^{m} \left(\sum_{j \in A \cap M_i} p_j \right)^2 + \frac{1}{2} \sum_{j \in A} p_j^2 \\
&\geq \frac{1}{2m} \left(\sum_{i=1}^{m} \sum_{j \in A \cap M_i} p_j \right)^2 + \frac{1}{2} \sum_{j \in A} p_j^2
\end{aligned}$$

□

Lemma 14.2:
If the numbers $C_1 \leq \ldots \leq C_n$ fulfill the first constraint (14.1), then with $J = \{1, \ldots, j\}$ we get

$$2C_j \geq \frac{1}{m} \sum_{k \in J} p_k \, .$$

Proof.

$$\begin{aligned}
& C_j \sum_{k \in J} p_k \geq \sum_{k \in J} p_k C_k \geq \frac{1}{2m} \left(\sum_{k \in J} p_k \right)^2 + \frac{1}{2} \sum_{k \in J} p_k^2 \geq \frac{1}{2m} \left(\sum_{k \in J} p_k \right)^2 \\
\Rightarrow \; & 2C_j \geq \frac{1}{m} \sum_{k \in J} p_k
\end{aligned}$$

□

Idea for an Approximation Algorithm

(A) Solve the LP. This gives an (LP) optimal solution C_j^{LP}. This can be done in polynomial time although we have exponentially many inequalities.

(B) We use the ordering $C_{j_1}^{LP} \leq \ldots \leq C_{j_n}^{LP}$ for a **job based priority list**

$$L = j_1 < j_2 < \ldots < j_n$$

We may start job j_k only after all $j_1, \ldots, j_{k-1}$have been started, i.e., we do list scheduling with conditions $S_{j_1} \leq \ldots \leq S_{j_n}$. This is different from priority lists as considered before.

(C) Use lemma 14.2 to prove a performance guarantee.

More on (B).

Theorem 14.3:

(1) Every job-based list scheduling rule defines a policy, called job-based list scheduling policy hereafter.

(2) Every job-based list scheduling policy is dominated by a preselective policy.

Proof.

(1) A job based list scheduling rule is clearly non-anticipative.

(2) Let $1 < 2 < \ldots < n$ be the priority list L and Π be the job based priority policy. The condition $S_1 \leq S_2 \leq \ldots \leq S_n$ implies that, for every forbidden set F the last job of F in L is selected as waiting job. The preselective policy Π^* with that selection S dominates Π because it does early start scheduling with respect to S and thus may violate $S_1 \leq \ldots \leq S_n$.

□

More on (C).

Lemma 14.4:
Let Π be a job-based priority policy with List $L = 1 < 2 < \ldots < n$. Define $C_j^{\Pi}(x) := \Pi[x](j) + x_j$ the completion time of job j with respect to Π and x. Then

$$C_j^{\Pi}(x) \leq \frac{1}{m} \sum_{k=1}^{j-1} x_k + x_j$$

holds for every x and thus we get in the stochastic case:

$$\mathbb{E}[C_j^{\Pi}] \leq \frac{1}{m} \sum_{k=1}^{j-1} \mathbb{E}[X_k] + \mathbb{E}[X_j]$$

Proof. Consider x fixed. When j is started the jobs $1, \ldots, j-1$ have already been started. The latest time by which a machine becomes available for j is

$$\frac{1}{m} \sum_{k=1}^{j-1} x_k \quad \text{(all machines are busy as long as possible)}$$

This gives

$$S_j \leq \frac{1}{m} \sum_{k=1}^{j-1} x_k \qquad \Rightarrow \quad C_j \leq \frac{1}{m} \sum_{k=1}^{j-1} x_k + x_j$$

Taking expectations gives the second inequality. □

Theorem 14.5:
Let $C_1^{LP} \leq C_2^{LP} \leq \ldots \leq C_n^{LP}$ be an optimal solution of the (LP) and (for fixed x) let $C_1, \ldots, C_n$ be the vector of completion times obtained by job-based list scheduling according to the list $L = 1 < 2 < \ldots < n$. Then we get

$$\sum_j w_j C_j \leq \left(3 - \frac{1}{m}\right) OPT$$

i.e., the algorithm for the LP-guided job-based priority scheduling is a $(3 - \frac{1}{m})$-approximation algorithm for the deterministic case.

Proof.

$$\begin{aligned} C_j &\overset{\text{lem14.4}}{\leq} \frac{1}{m}\sum_{k=1}^{j-1} x_k + x_j && = \frac{1}{m}\sum_{k=1}^{j} + \frac{m-1}{m}x_j \\ &\overset{\text{lem14.2}}{\leq} 2C_j^{LP} + \frac{m-1}{m}x_j && \leq \left(3 - \frac{1}{m}\right) C_j^{LP} \\ \Rightarrow \sum_j w_j C_j &\leq \left(3 - \frac{1}{m}\right) \sum_j w_j C_j^{LP} && \leq \left(3 - \frac{1}{m}\right) OPT \end{aligned}$$

□

More on (A)
We can solve the (LP) in polynomial time if the separation problem for (14.1) and (14.2) can be solve in polynomial time. This is trivial for (14.2). So let us show this for (14.1).
For a given vector $(C_1, \ldots, C_n) \in \mathbb{R}^n$ and $A \subseteq V$ we define the violation as

$$v(A) := \frac{1}{2m}\left(\sum_{j\in A} p_j\right)^2 + \frac{1}{2}\sum_{j\in A} p_j^2 - \sum_{j\in A} p_j C_j$$

Lemma 14.6:
Let A maximize the violation. Then $k \in A$ if and only if

$$C_k - \frac{1}{2}p_k < \frac{1}{m}\sum_{j\in A} p_j$$

Proof. First we get for $k \in A$

$$v(A\backslash\{k\}) = v(A) - p_k\left(\frac{1}{m}\sum_{j\in A} p_j + \frac{m-1}{2m}p_k - C_k\right) \tag{14.3}$$

and for $k \notin A$ we get

$$v(A \cup \{k\}) = v(A) + p_k\left(\frac{1}{m}\sum_{j\in A} p_j + \frac{m+1}{2m}p_k - C_k\right) \tag{14.4}$$

Now let A maximize the violation then we can conclude

$k \in A$ implies $v(A\backslash\{k\}) \leq v(A)$. Using (14.3) we can calculate

$$C_k \leq \frac{1}{m}\sum_{j\in A} p_j + \frac{m-1}{2m}p_k < \frac{1}{m}\sum_{j\in A} p_j + \frac{1}{2}p_k$$

$k \notin A$ implies $v(A \cup \{k\}) \leq v(A)$. Using (**??**) we can calculate

$$\frac{1}{m}\sum_{j\in A} p_j + \frac{m+1}{2m}p_k \leq C_k$$

□

Separation Algorithm

(1) Sort the jobs with respect to increasing $C_j - \frac{1}{2}p_j$ values. Let $1 < 2 < \ldots < n$ be this ordering.

(2) The set A with maximum violation is an initial segment $J = \{1, 2, \ldots, j\}$ of this ordering.

(3) Check the initial segments of the ordering for violation.

Proof of (2). Let A maximize the violation and $i \in A$. We show that $k \in A$ for every $k \leq i$. So let $i \in A$. This implies with lemma 14.6

$$C_i - \frac{1}{2}p_i < \frac{1}{m}\sum_{j \in A} p_j$$

Since $k \leq i$ the ordering and lemma 14.6 imply

$$C_k - \frac{1}{2}p_k \leq C_i - \frac{1}{2}p_i < \frac{1}{m}\sum_{j \in A} p_j \Rightarrow k \in A$$

□

Checking (3) clearly requires only polynomial time.

We now consider the stochastic case with independent processing times. Let us look at an LP-based approach. Consider the achievable region

$$\left\{(\mathbb{E}[C_1^{\Pi}], \ldots, \mathbb{E}[C_n^{\Pi}]) \in \mathbb{R}^n \,\middle|\, \Pi \text{ policy}\right\}$$

Find a polyhedral relaxation $\boldsymbol{P}$. Solve the linear program

$$\begin{aligned} \min \quad & \sum_j w_j C_j^{LP} \\ \text{s.t.} \quad & C^{LP} \in \boldsymbol{P} \end{aligned}$$

Then use the list $L : i_1 < i_2 < \ldots < i_n$ defined by $C_{i_1}^{LP} \leq C_{i_2}^{LP} \leq \ldots \leq C_{i_n}^{LP}$ as list for priority policies.
To find the polyhedral relaxation we generalize the valid inequalities from deterministic scheduling according to [HSDW97].

$$\sum_{j \in A} \mathbb{E}[X_j]\mathbb{E}[C_j^{\Pi}] \geq \frac{1}{2m}\left(\sum_{j \in A} \mathbb{E}[X_j]\right)^2 + \frac{1}{2}\sum_{j \in A} \mathbb{E}[X_j]^2 - \frac{m-1}{2m}\sum_{j \in A} Var[X_j]$$
$$\forall A \subseteq \{1, \ldots, n\} \text{ and all policies } \Pi \quad (14.5)$$

Lemma 14.7:
Every policy Π *(for an* m*-machine problem without precedence constraints) fulfills* (14.5).

Proof. Consider a fixed realization $x = (x_1, \dots, x_n)$. Then lemma 14.1 gives

$$\sum_{j \in A} x_j C_j^{\Pi}(x) \geq \frac{1}{2m} \left(\sum_{j \in A} x_j \right)^2 + \frac{1}{2} \sum_{j \in A} x_j^2 \quad \forall A \subseteq V$$

We rewrite this in terms of start times $S_j^{\Pi}(x) = C_j^{\Pi} - x_j$ and get

$$\sum_{j \in A} x_j S_j^{\Pi}(x) \geq \frac{1}{2m} \left(\sum_{\substack{i,j \in A \\ i \neq j}} x_i x_j \right) - \frac{m-1}{2m} \sum_{j \in A} x_j^2 \tag{14.6}$$

The S_j^{Π} and X_j are stochastically independent because Π is a policy and thus non-anticipative. Furthermore we can now conclude from stochastics

$$\mathbb{E}[X_j \cdot S_j^{\Pi}] = \mathbb{E}[X_j] \cdot \mathbb{E}[S_j^{\Pi}] \qquad Var[X_j] = \mathbb{E}[X_j^2] - \mathbb{E}[X_j]^2 \tag{14.7}$$

We now take expectation in (14.6) and use linearity of expectation, (14.7) and independency of processing times to get

$$\begin{aligned}
\sum_{j \in A} \mathbb{E}[X_j \cdot S_j^{\Pi}] &\overset{(14.7)}{=} \sum_{j \in A} \mathbb{E}[X_j] \cdot \mathbb{E}[S_j^{\Pi}] \\
&\geq \frac{1}{2m} \sum_{\substack{i,j \in A \\ i \neq j}} \mathbb{E}[X_i X_j] - \frac{m-1}{2m} \sum_{j \in A} \mathbb{E}[X_j^2] \\
&= \frac{1}{2m} \sum_{\substack{i,j \in A \\ i \neq j}} \mathbb{E}[X_i]\mathbb{E}[X_j] - \frac{m-1}{2m} \sum_{j \in A} \mathbb{E}[X_j^2] \\
&= \frac{1}{2m} \left(\sum_{j \in A} \mathbb{E}[X_j] \right)^2 - \frac{1}{2m} \sum_{j \in A} \mathbb{E}[X_j]^2 - \frac{m-1}{2m} \sum_{j \in A} \mathbb{E}[X_j^2]
\end{aligned}$$

Now looking an the two last sums

$$\begin{aligned}
&- \frac{1}{2m} \sum_{j \in A} \mathbb{E}[X_j]^2 - \frac{m-1}{2m} \sum_{j \in A} \mathbb{E}[X_j^2] \\
&= \frac{m-1}{2m} \sum_{j \in A} \mathbb{E}[X_j]^2 - \frac{m-1}{2m} \sum_{j \in A} \mathbb{E}[X_j^2] - \frac{m-1}{2m} \sum_{j \in A} \mathbb{E}[X_j]^2 - \frac{1}{2m} \sum_{j \in A} \mathbb{E}[X_j]^2 \\
&\overset{(14.7)}{=} -\frac{m-1}{2m} \sum_{j \in A} Var[X_j] - \frac{1}{2} \sum_{j \in A} \mathbb{E}[X_j]^2
\end{aligned}$$

Therefore we get

$$\sum_{j \in A} \mathbb{E}[X_j] \cdot \mathbb{E}[C_j^{\Pi}] \geq \frac{1}{2m} \left(\sum_{j \in A} \mathbb{E}[X_j] \right)^2 + \frac{1}{2} \sum_{j \in A} \mathbb{E}[X_j]^2 - \frac{m-1}{2m} \sum_{j \in A} Var[X_j]$$

by adding $\sum \mathbb{E}[X_j]^2$ on both sides. □

Note that we got a similar inequality to the deterministic case, except for the variances.

Definition 14.8:
For a random variabe X_j we define the ***coefficient of variation*** *of X_j as*

$$CV[X_j] := \frac{Var[X_j]}{\mathbb{E}[X_j]^2}$$

This coefficient is ≤ 1 for all NBUE (New Better than Used in Expectation) distributions (e.g. exponential, uniform). The NBUE property reads in mathematical terms as:

$$\mathbb{E}[X_j - t | X_j > t] \leq \mathbb{E}[X_j] \quad \forall t > 0$$

Discrete and multi-modal distributions are not NBUE.

Lemma 14.9:
Assume that $CV[X_j] \leq \Delta$ holds for all j, then this implies

$$\sum_{j \in A} \mathbb{E}[X_j]\mathbb{E}[C_j^{\Pi}] \geq \frac{1}{2m} \left[\left(\sum_{j \in A} \mathbb{E}[X_j] \right)^2 + \sum_{j \in A} \mathbb{E}[X_j]^2 \right] - \frac{(m-1)(\Delta - 1)}{2m} \sum_{j \in A} \mathbb{E}[X_j]^2 \,. \quad (14.8)$$

Proof. Just calculation. □

Lemma 14.10:
Let $C = (C_{j_1}, \ldots, C_{j_n}) \in \mathbb{R}^n$ fulfill (14.8). Assume $C_1 \leq \ldots \leq C_n$ and $C_j \geq \mathbb{E}[X_j]$ then

$$\frac{1}{m} \sum_{k=1}^{j} \mathbb{E}[X_k] \leq \left(1 + \max \left\{ 1, \frac{m-1}{m} \Delta \right\} \right) C_j \qquad \forall j$$

Proof. First consider

$$\begin{aligned} C_j \sum_{k=1}^{j} \mathbb{E}[X_k] &\geq \sum_{k=1}^{j} \mathbb{E}[X_k] C_k \\ &\geq \frac{1}{2m} \left(\sum_{k=1}^{j} \mathbb{E}[X_k] \right)^2 + \frac{m - \Delta(m-1)}{2m} \sum_{k=1}^{j} \mathbb{E}[X_k]^2 \end{aligned}$$

Then we divide by $\sum \mathbb{E}[X_k]$ and get

$$C_j \geq \frac{1}{2m} \sum_{k=1}^{j} \mathbb{E}[X_k] + \frac{m - \Delta(m-1)}{2m} \cdot \frac{\sum_k \mathbb{E}[X_k]^2}{\sum_k \mathbb{E}[X_k]}$$

Case 1 $\Delta \leq \frac{m}{m-1}$ Then we get

$$C_j \geq \frac{1}{2m} \sum_{k=1}^{j} \mathbb{E}[X_k]$$

Case 2 $\Delta > \frac{m}{m-1}$ Now the second term is negative. We use that $C_j \geq C_k \geq \mathbb{E}[X_k]$ for $k = 1, \ldots, j$ and get

$$C_j \geq \max_{k=1,\ldots,j} \mathbb{E}[X_k] \geq (\sum_{k=1}^{j} \mathbb{E}[X_k]^2) \cdot (\sum_{k=1}^{j} \mathbb{E}[X_k])^{-1} .$$

Now let the maximum be attained at index i_0, i.e. $\mathbb{E}[X_{i_0}] = \max \mathbb{E}[X_k]$ and thus

$$\sum_{k=1}^{j} \mathbb{E}[X_k]^2 \leq \mathbb{E}[X_{i_0}] \sum_{k=1}^{j} \mathbb{E}[X_k] .$$

Therefore we can conclude

$$C_j \geq \frac{1}{2m} \sum_{k=1}^{j} \mathbb{E}[X_k] + \frac{m - \Delta(m-1)}{2m} C_j$$

Now case 1 and case 2 imply the claim. □

Remark:
The previous lemma implies that the average load of the machines of an initial segment of $\{1, \ldots n\}$ is less or equal to a constant times the completion of the last job. So this is the stochastic counterpart of lemma 14.1. △

Theorem 14.11:
Let Π *be the job based policy induced by the (LP)*

$$\begin{aligned} &\min \sum_j w_j C_j^{LP} \\ &s.t. \ (14.8) \\ &\quad C_j^{LP} \geq \mathbb{E}[X_j] \quad \forall j \end{aligned}$$

Then Π *is a* $(2 + \max\{1, \frac{m-1}{m}\Delta\})$*-approximation.*

Proof.
Let $C_1^{LP} \leq \ldots \leq C_n^{LP}$ be an optimal solution of the LP and let $L := 1 < 2 < \ldots < n$ be a priority list and Π be the induced policy. Then we can estimate

$$\begin{aligned} \mathbb{E}[C_j^{\Pi}] &\leq \frac{1}{m} \sum_{k=1}^{j-1} \mathbb{E}[X_k] + \mathbb{E}[X_j] \\ &= \frac{1}{m} \sum_{k=1}^{j} \mathbb{E}[X_k] + \frac{m-1}{m} \mathbb{E}[X_j] \\ &\overset{??}{\leq} \left(1 + \max\left\{1, \frac{m-1}{m}\Delta\right\}\right) C_j^{LP} + \frac{m-1}{m} C_j^{LP} \\ &\leq \left(2 + \max\left\{1, \frac{m-1}{m}\Delta\right\} - \frac{1}{m}\right) C_j^{LP} \end{aligned}$$

This now implies

$$\mathbb{E}[\sum_{j=1}^{n} w_j C_j^{\Pi}] = \sum_{j=1}^{n} w_j \mathbb{E}[C_j^{\Pi}] \leq \left(2 - \frac{1}{m} + \max\left\{\frac{m-1}{m}\Delta\right\}\right) \sum_{j=1}^{n} w_j C_j^{LP}$$
$$\leq \underbrace{\left(2 - \frac{1}{m} + \max\left\{\frac{m-1}{m}\Delta\right\}\right)}_{\alpha_m} \cdot OPT$$

□

Remark:

(1) The LP can be solved in polynomial time (see Theorem 14.12).

(2) The WSEPT[3] first rule, i.e.

$$\frac{w_{j_1}}{\mathbb{E}[X_{j_1}]} \geq \ldots \geq \frac{w_{j_k}}{\mathbb{E}[X_{j_k}]}$$

leads to a guarantee of

$$1 + \frac{(\Delta+1)(m-1)}{2m} \approx \alpha_m - 1$$

(3) These results can be generalized to problems with release dates $r_j \geq 0$. Using a job based priority policy we then get an $\alpha_m + 1$ guarantee.

△

Theorem 14.12:
Assume that the jobs are ordered according to WSEPT, meaning

$$\frac{w_1}{\mathbb{E}[X_1]} \geq \ldots \geq \frac{w_n}{\mathbb{E}[X_n]}$$

Then the LP has the optimal solution

$$C_j^{LP} = \frac{1}{m} \sum_{k=1}^{j} \mathbb{E}[X_k] - \frac{(\Delta-1)(m-1)}{2m} \mathbb{E}[X_j] \qquad j = 1, \ldots, n$$

for fixed $\Delta \geq 1$, $m \in \mathbb{N}$

Proof. Idea:
Let $f(A)$ be the right hand side of the inequalities of (14.8) for $A \subseteq V$. Then $f : 2^V \to \mathbb{R}$ is *supermodular*, i.e.

$$f(A \cup B) + f(A \cap B) \geq f(A) + f(B) \quad \forall A, B \subseteq V$$
$$\Leftrightarrow f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) \quad \forall A \subset B \subset V\,,\, x \in V \backslash B$$

Therefore the polyhedron defined by the LP (14.8) is a supermodular polyhedron. Thus the optimal solution can be obtained by Edmond's Greedy Algorithm for such supermodular polyhedra. □

[3] Weighted Shortest Expected Processing Time

Exercises

14.1 Show that WSEPT leads to a $1 + \frac{(\Delta+1)(m-1)}{2m}$ approximation (on m machines with no precedence constraints).

14.2 Generalize the results to the case with release dates. (Use a job based priority rule.)

14.3 Show that WSEPT may be arbitrary bad for release dates.

Consequences

(A) WSEPT is an optimal policy for $m = 1$.

(B) The guarantee of WSEPT is better than the one given in theorem 14.11 and is the best known for problems without release dates.

(C) Theorem 14.11, however, can be generalized to release dates and gives a guarantee of $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$.

Theorem 14.13: Master Thesis of B. Labonté 2013
The term Δ $(=: k)$ is essential, i.e., there are instances $I(m, n, k)$ such that

$$\frac{WSEPT(I(m,n,k))}{OPT(I(m,n,k))} \in \Omega(k^{\frac{1}{4}}) \ .$$

Where we have n jobs, m machines and $k = \Delta \geq CV(X_j)$.

Sketch of Proof. The instances $I(m, n, k)$ are constructed as follows.
We take m deterministic jobs with weights $w_j = 1$ and processing times $x_j = r > 0$ and $n - m$ stochastic jobs with weights $w_j = \sqrt{\frac{m}{m-1}}$ and processing times

$$X_j = \begin{cases} rw_jk & \text{with probability } \frac{1}{k} \\ 0 & \text{with probability } 1 - \frac{1}{k} \end{cases}$$

Thus we have

$$CV(X_j) \leq k - 1 \quad \text{and} \quad \frac{w_j}{\mathbb{E}[X_j]} = \frac{1}{r} \quad \forall j$$

So every order is WSEPT. We use two different policies based on different job orderings:

$$\Pi_{\text{det-first}} \qquad \text{and} \qquad \Pi_{\text{stoch-first}} \ .$$

Then we obviously have

$$\frac{\mathbb{E}[\kappa^{\Pi}_{\text{det}}]}{\mathbb{E}[\kappa^{\Pi}_{OPT}]} \geq \frac{\mathbb{E}[\kappa^{\Pi}_{\text{det}}]}{\mathbb{E}[\kappa^{\Pi}_{\text{stoch}}]}$$

and furthermore for $n = m + \lfloor\sqrt{k}\rfloor$

$$\mathbb{E}[\kappa^{\Pi}_{\text{det}}] \geq \left(1 - \frac{1}{k}\right)^{n-m} (m \cdot r + (n-m)r \cdot w) \quad \in \Omega(k^{\frac{1}{4}})$$

and $\mathbb{E}[\kappa^{\Pi}_{\text{stoch}}] = O(1)$. □

15 Stochastic Online Scheduling for Weighted Completion Times

We follow some results of [MUV06]. Consider a 2 phase model for the weighted sum of completion times on m machines.

- jobs arrive online and must be assigned to machines immediately. There is an unknown number of jobs which have a random processing time each.
- *on the next day* the jobs are scheduled on the assigned machines in the *expected performance* model. The number of jobs is now known. (We do this optimally on every machine with WSEPT)
- view this as a scheduling policy and analyze with respect to expected performance.

Now we want to develop an algorithm called Min-Increase.

- Assign jobs to a machine such that $\sum w_j C_j$ based on $\mathbb{E}[X_j]$ has minimum increase and can be done in polynomial time.
- Min-Increase matches best known bounds of previous model. This can be done even better for NBUE processing times and release dates.
- Needs LP-based lower bounds in analysis but not for defining the policy.
- This was the first combinatorial approximation algorithm for release dates.

Let us introduce some notations using the priority orders from WSEPT

- $j \to i$ if job j is assigned to machine i
- $H(j) = \{k \in V \mid \text{higher priority as } j\} \cup \{j\}$
- $L(j) = V \backslash H(j)$ lower priority jobs
- $k < j$ corresponds to k arrives before j
- We tie break according to the incoming order.

Algorithm Min-Increase

(1) Upon arrival of job j, assign it to machine i that minimizes

$$z(j,i) := w_j \sum_{\substack{k \in H(j) \\ k<j \\ k \to i}} \mathbb{E}[X_k] + \mathbb{E}[X_j] \sum_{\substack{k \in L(j) \\ k<j \\ k \to i}} w_k + w_j \mathbb{E}[X_j]$$

(2) In the scheduling phase, schedule the jobs on every machine according to WSEPT (optimal per machine).

Lemma 15.1:
The value $z(j,i)$ is the increase of $\sum_l w_l \mathbb{E}[C_l]$ on machine i when j is assigned to that machine and jobs are scheduled by WSEPT.

Proof.
Consider machine i

The first part doesn't change. The expected completion time of j gets weight w_j. All jobs after j are delayed by the expected processing time of j. □

Lemma 15.2:
The expected value of the policy induced by Min-Increase is equal to the sum over the minimum increase for assigning the jobs on arrival, i.e.

$$\mathbb{E}\left[\sum_{j=1}^{n} w_j C_j^{MI}\right] = \sum_{j=1}^{n} \min_i z(j,i)$$

Proof. Let us abbreviate $C_j := C_j^{MI}$ for now. We partition the jobs as follows

$$\mathbb{E}\left[\sum_{j=1}^{n} w_j C_j\right] = \sum_{j=1}^{n} w_j \sum_{\substack{k \in H(j) \\ k \to i_j \\ k<j}} \mathbb{E}[X_k] + \sum_{j=1}^{n} \sum_{\substack{k \in H(j) \\ k \to i_j \\ k>j}} \mathbb{E}[X_k] + \sum_{j=1}^{n} w_j \mathbb{E}[X_j]$$

into the jobs that arrive before and after j. The index i_j denotes the machine to which j is assigned.

Claim:

$$\sum_{j=1}^{n} w_j \sum_{\substack{k \in H(j) \\ k>j \\ k \to i_j}} \mathbb{E}[X_j] = \sum_{j=1}^{n} \mathbb{E}[X_j] \sum_{\substack{k \in L(j) \\ k<j \\ k \to i_j}} w_k$$

Sketch of proof. We count the jobs rowwise and columnwise in a $2-D$ Gantt chart. Now the left hand side corresponds to rowwise counting and the right hand side to columnwise counting. □

The claim implies

$$\mathbb{E}\left[\sum_{j=1}^{n} w_j C_j\right] = \sum_{j=1}^{n} \Big(w_j \sum_{\substack{k \in H(j) \\ k<j \\ k \to i_j}} \mathbb{E}[X_k] + \mathbb{E}[X_j] \sum_{\substack{k \in L(j) \\ k<j \\ k \to i_j}} w_k + w_j \mathbb{E}[X_j] \Big) = \sum_{j=1}^{n} \min_i z(j,i)$$

□

Theorem 15.3:
Let $CV[X_j] \leq \Delta$. Then Min-Increase is a ϱ-approximation algorithm with

$$\varrho = 1 + \frac{(m-1)(\Delta+1)}{2m}$$

Proof. Using the lemma 15.2 and 15.1, i.e. the claim from the proof above, we can estimate

$$\begin{aligned}
\mathbb{E}[MI(I)] &= \sum_{j=1}^{n} \min_i z(j,i) \\
&= \sum_{j=1}^{n} \min_i \Big\{ w_j \sum_{\substack{k \in H(j) \\ k<j \\ k \to i}} \mathbb{E}[X_k] + \mathbb{E}[X_j] \sum_{\substack{k \in L(j) \\ k<j \\ k \to i}} w_k + w_j \mathbb{E}[X_j] \Big\} \\
&= \sum_{j=1}^{n} \min_i \Big\{ w_j \sum_{\substack{k \in H(j) \\ k<j \\ k \to i}} \mathbb{E}[X_k] + \mathbb{E}[X_j] \sum_{\substack{k \in L(j) \\ k<j \\ k \to i}} w_k \Big\} + \sum_{j=1}^{n} w_j \mathbb{E}[X_j]
\end{aligned}$$

The minimum of m elements is less than or equal to the arithmetic mean of these m elements. Therefore we get

$$\begin{aligned}
\mathbb{E}[MI(I)] &\leq \sum_{j=1}^{n} \frac{1}{m} \sum_{i=1}^{m} \Big\{ w_j \sum_{\substack{k \in H(j) \\ k<j \\ k \to i_j}} \mathbb{E}[X_k] + \mathbb{E}[X_j] \sum_{\substack{k \in L(j) \\ k<j \\ k \to i_j}} w_k \Big\} + \sum_{j=1}^{n} w_j \mathbb{E}[X_j] \\
&= \sum_{i=1}^{m} \frac{1}{m} \Big\{ \sum_{j=1}^{n} w_j \sum_{\substack{k \in H(j) \\ k<j \\ k \to i}} \mathbb{E}[X_k] + \sum_{j=1}^{n} \mathbb{E}[X_j] \sum_{\substack{k \in L(j) \\ k<j \\ k \to i}} \Big\} + \sum_{j=1}^{n} w_j \mathbb{E}[X_j] \\
&= \sum_{i=1}^{m} \frac{1}{m} \sum_{j=1}^{n} \sum_{\substack{k \in H(j) \\ k \neq j \\ k \to i}} \mathbb{E}[X_k] + \sum_{j=1}^{n} w_j \mathbb{E}[X_j] \\
&= \sum_{j=1}^{n} \frac{1}{m} w_j \sum_{i=1}^{m} \sum_{\substack{k \in H(j) \\ k \neq j \\ k \to i}} \mathbb{E}[X_k] + \sum_{j=1}^{n} w_j \mathbb{E}[X_j] \\
&= \sum_{j=1}^{n} \frac{1}{m} w_j \sum_{k \in H(j)} \mathbb{E}[X_k] + \frac{m-1}{m} \sum_{j=1}^{n} w_j \mathbb{E}[X_j]
\end{aligned}$$

Now we use the inequality (**??**) from lemma 15.4 and conclude

$$\begin{aligned}
&\Rightarrow \mathbb{E}[MI(I)] \leq \mathbb{E}[OPT(I)] + \Big(\frac{(m-1)(\Delta-1)}{2m} + \frac{m-1}{m} \Big) \sum_{j=1}^{n} w_j \mathbb{E}[X_j] \\
&\Rightarrow \mathbb{E}[MI(I)] \leq \varrho \cdot \mathbb{E}[OPT(I)]
\end{aligned}$$

□

Lemma 15.4:
Consider priorities with respect to non-increasing $\frac{w_j}{\mathbb{E}[X_j]}$ *values. Then the following inequality holds*

$$\mathbb{E}[OPT(I)] \geq \sum_{j=1}^{n} w_j \frac{1}{m} \sum_{k \in H(j)} \mathbb{E}[X_k] - \frac{(m-1)(\Delta-1)}{2m} \sum_{j=1}^{n} w_j \mathbb{E}[X_j] \tag{15.1}$$

Proof. Recall theorem 14.11, i.e. an optimal solution to the LP defined by inequality (14.8) and $C_j^{LP} \geq \mathbb{E}[X_j]$ in the section 14 is given by

$$C_j^{LP} = \frac{1}{m} \sum_{k \in H(j)} \mathbb{E}[X_k] - \frac{(m-1)(\Delta-1)}{2m} \mathbb{E}[X_j]$$

$$\Rightarrow \sum_{j=1}^{n} w_j C_j^{LP} = \sum_{j=1}^{n} w_j \frac{1}{m} \sum_{k \in H(j)} \mathbb{E}[X_k] - \frac{(m-1)(\Delta-1)}{2m} \mathbb{E}[X_j]$$

and thus

$$\mathbb{E}[OPT(I)] \geq \sum_{j=1}^{n} w_j C_j^{LP}$$

□

Remark:(1) The performance guarantee of Min-Increase matches the best known in the offline setting, but does not require the knowledge of all jobs and their expected processing times in advance, but only when they arrive.

(2) WSEPT and Min-Increase produce different schedules in general.

(3) The lower bound of theorem 14.12 applies to Min-Increase.

△

Exercises

15.1 Show by example that WSEPT and Min-Increase generate different schedules.

We want to derive a simple lower bound for Min-Increase

Example 22 :
Asymptotically we get

$$\frac{\mathbb{E}[MI(I)]}{\mathbb{E}[OPT(I)]} \geq \frac{1}{2}$$

Let an instance I be given by $n-1$ deterministic jobs $\{1, \ldots, n-1\}$ with processing times $x_j = 1$ and weights $w_j = 1$ and 1 stochastic job n with a 2-point distribution

$$X_n := \begin{cases} 1 & \text{with probability } 1 - \frac{2}{n} \\ \frac{n^2}{4} & \text{with probability} \frac{2}{n} \end{cases} \qquad w_n = \mathbb{E}[X_n] = 1 - \frac{2}{n} + \frac{n}{2}$$

We consider 2 machines and the jobs arrive 1 to n. Therefor Min-Increase assigns the $n-1$ deterministic jobs first. Then job n is assigned to the less busy machine. This gives

$$\mathbb{E}[MI(I)] = \mathbb{E}\left[\sum_j w_j C_j\right] = 3\frac{n^2}{4} + o(n^2)$$

An optimal policy starts only job n and fixes $t = 1$ as next tentative decision time. If job n takes longer than 1 put all remaining jobs on the other machine (else split them on both). This gives

$$\mathbb{E}[OPT(I)] = \frac{n^2}{2} + o(n^2)$$

So the ratio goes to $\frac{1}{2}$ for n to infinity. ♢

16 Evaluating the Distribution of the Objective Value of a Policy

We will do this in two steps.

(1) Determine a policy Π (for a distribution of processing times Q and a cost function κ).

(2) Evaluate its cost distribution Q_κ^Π.

This is well investigated for ES-policies, and $\kappa = C_{\max}$ (makespan). Ideal we want a distribution function F of $C_{\max}$ or at least important percentiles.

Hagstrom '88 [Hag88]

MEAN For a stochastic project network (V, G, Q) (defines ES-policy) with discrete independent processing times we want to calculate the expected makespan.

DF For a given stochastic network with discrete independent processing time and some time t we want to know the probability of (makespan $\leq t$).

Theorem 16.1: Hagstrom

(1) MEAN and DF are polynomially equivalent for 2-point distrbutions.

(2) MEAN and the 2-point version of DF are #P-complete.

(3) Unless $P = NP$, MEAN and DF cannot be solved in time polynomial in the number of values in the (also) discrete distribution that the makespan attains.

Definition 16.2:
***Number-**P, $\#P$, is a complexity class for counting problems. A decision problem is in $\#P$ if one can compute in non-deterministic polynomial time the number of Yes-answers for every instance I of that problem. So a problem is in $\#P$ if and only if there is a polynomial p and a non-deterministic algorithm A such that for all $I \in \mathcal{P}$ $A(I)$ gives the number of Yes-answers to instance I in $p(< I >)$[4]. We say that A solves the counting problem.*

Example 23 :

(1) We know SAT is NP and models the question: Does a CNF formula have a fulfilling truth assignment?
Now #SAT is $\#P$ and models the question: How many fulfilling assignments has a CNF formula?

(2) As above for the NP problem: Does a graph G have a Hamiltonian Cycle? The counting version: How many Hamiltonian Cycles does a graph have? is $\#P$.

[4] $< I >$ denotes the encoding length.

(3) The linear extension problem is in P (Does a partial order have a linear extension?). The problem: How many linear extension does a partial order have? is $\#P$.

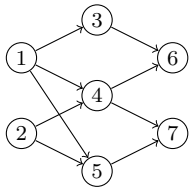
♢

So problems in P can also lead to $\#P$ problems. Similar to NP-complete one defines $\#P$-complete. We use Turing reduction instead of polynomial or Karp reduction.

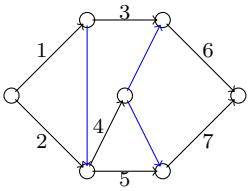
Definition 16.3:
A problem $\mathcal{P} \in \#P$ is $\#P$-complete if and only if there is a (polynomial) Turing reduction from $\mathcal{P}' \in \#P$-complete to $\mathcal{P}$ such that $\mathcal{P}' \to \mathcal{P}$. So counting on $\mathcal{P}$ solves that counting problem in $\mathcal{P}'$ in polynomial time.

For the proof of Hagstroms theorem 16.1 we need a different representation of partial orders. So far we considered jobs as vertices and G as transitive reduction in a node diagram. Now we use an arc diagram and consider jobs as arcs and a digraph D with a unique source s and a unique sink t. We may need artificial dummy jobs to represent precedence constraints correctly. More precisely $(i, j) \in E(G)$ if and only if there is a directed path from the tail of i to the head of j.

Example 24 :
Node diagram: The jobs are nodes of the digraph G.



Arc diagram: The jobs are arcs of digraph D.



The blue arcs are dummy jobs. The makespan corresponds the length of a longest path. ♢

Remark:

- An arc diagram enables us to use standard graph algorithms directly.
- The resulting arc diagrams are not unique.
- Constructing an arc diagram with a minimum number of dummy arcs is NP-hard.

△

Standard Construction
Start from G. Uses edges of G as dummy arcs. Expand every node of G to an arc and contract afterwards. Identify the tail nodes of all minimal jobs and the head nodes of all maximal jobs and contract *superflous* dummy arcs.

Proof of Theorem 16.1. We will only proof claim for the 2-point version of DF. We will use a reduction from RELIABILITY which is $\#P$-complete.

Reliability
For a given directed acyclic $s-t$ graph D with independent failure probabilities q_j on arc j we want to know the probability of the event that there exists an $s-t$ path without failures (the reliability of D).
Let I be an instance of Reliability. Construct an instance I' of DF such that for some appropriate $L \in \mathbb{N}$ we have

$$\text{reliability of } D = 1 - \mathbb{P}(\{C_{\max} \leq L-1\}) = 1 - F(L-1)$$

So we can compute the reliabilty from one evaluation of F.
Construction of I'
Take D as an arc diagram of G. Every job j has processing time

$$X_j = \begin{cases} 0 & \text{with } \mathbb{P} = q_j \\ p_j & \text{with } \mathbb{P} = 1 - q_j \end{cases}$$

with $p_j \in \mathbb{N}$ such that all $s-t$ paths have the same length L. This can be done in polynomial time. Now we get

$$\begin{aligned} \mathbb{P}(\text{some path is reliable}) &= 1 - \mathbb{P}(\text{all paths fail}) \\ &= 1 - \mathbb{P}(\cap\{P_i \text{ fails}\}) \\ &= 1 - \mathbb{P}(\cap\{\text{some arc in } P_i \text{ fails}\}) \\ &= 1 - \mathbb{P}(\cap\{P_i \text{ has length } < L\}) \\ &= 1 - \mathbb{P}(\cap\{C_{\max} \leq L-1\}) \\ &= 1 - F(L-1) \end{aligned}$$

This shows $\#P$-hardness. See [Hag88] for DF in $\#P$. □

Exercises

16.1 Characterization of partial orders that have an arc diagram without dummy arcs:
A partial order is ***N*-free** if its Hasse diagram does not contain an N (viewed as nodes and arcs) as subgraph. Mathematically

$$\forall i,j \quad \text{ImPred}(i) = \text{ImPred}(j) \quad \text{or } \text{ImPred}(i) \cap \text{ImPred}(j) = \emptyset$$

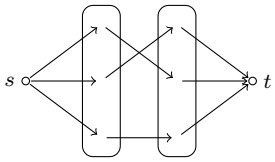
A partial order has the ***CAC*-property** if every maximal chain and every maximal antichain have a non-empty intersection (also called the chain-antichain property). Show that the following are equivalent for a partial order $G = (V, E)$

(1) G has an arc-diagram without dummy arcs.

(2) G is N-free.

(3) G has the CAC property.

16.2 Strengthen the proof of theorem 16.1 to processing times $X_j \in \{0, 1\}$. Hint: Use the fact that Reliability is already $\#P$-complete for $s-t$ dags of the form (bipartite)



16.3 Show that the 2-point versions of MEAN and DF with $X_j \in \{0, 1\}$ are polynomially equivalent.

17 Bounding the Distribution Function of the Makespan

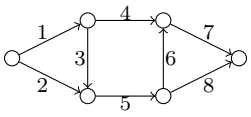
One approach is simulation. There one needs exact information about the processing time distribution and it is difficult to model stochastic dependencies. Therefore we take another approach here. We will compute bounds. This is closely related to network algorithms.

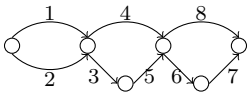
Definition 17.1:
*Let N, N' be stochastic project networks. We say N is a **chain minor** of N' if*

- *every job of N is represented by one or several copies in N' (but N' may not contain more than those copies).*
- *every maximal chain of N is contained in a maximal chain of N' by taking appropriate copies of jobs and the order of jobs (copies) along the chain is preserved.*

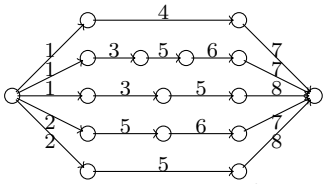
Example 25 :

We want to represent the simple network N two different chain majors N_1, N_2.





N_1 has no copies of jobs but proper containment of chains.



N_2 is an extreme case. It is a parallel decomposition of all maximal chains.

These are just some examples. ♢

Definition 17.2:
*Let X_1, X_2 be real valued random variables with distributions P_1, P_2 and distribution functions F_1, F_2. Then X_1, P_1 is **stochastically smaller** than X_2, P_2 if $F_1 \geq F_2$ holds pointwise, i.e.*

$$\forall t \quad \mathbb{P}(\{X_1 \leq t\}) \geq \mathbb{P}(\{X_2 \leq t\})$$
$$\Leftrightarrow$$
$$\int f \, dP_1 \leq \int f \, dP_2 \quad \forall f : \mathbb{R} \to \mathbb{R} \text{ non-decreasing}$$

Intuitively: X_1 is smaller than X_2 with higher probability.

Theorem 17.3:
Let N be a chain minor of N'. The copies of j in N' shall have the same processing time distributions as in N. But consider all processing time distributions in N' as

independent. Then the makespan of N is stochastically smaller than the makespan of N'.

$$Q_{C_{\max}(N)} \leq_{st} Q_{C_{\max}(N')}$$

Remark:
We will use the bounding principle to *sandwich* the unknown distribution function F of $C_{\max}$. △

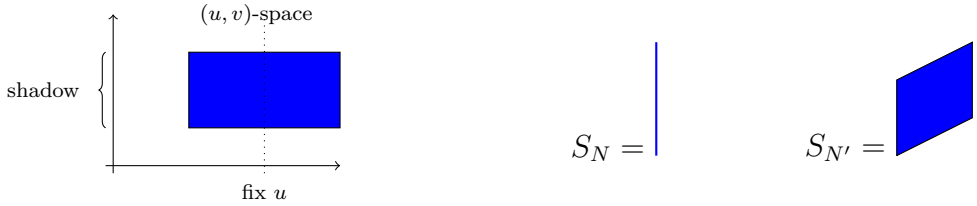
Proof. Let N be a chain minor of N'. Then N' can be obtained by a finite sequence $N = N_0 \prec N_1 \prec \ldots \prec N_k = N'$ such that N_i is obtained from N_{i-1} by one of the following operations.

- Duplicating one job only.
- N_i as partial order is an extension of N_{i-1} and no duplication is made.

The second operation is trivial for stochastically larger. So it suffices to show the claim for duplicating one job.
We prove this for an example. This proof can easily be generalized (it's quite generic). So we need to show

$$\mathbb{P}(C_{\max}(N) \leq t) \geq \mathbb{P}(C_{\max}(N') \leq t) \quad \forall t \, .$$

We want to compare a set in $\mathbb{R}^m$ and a set in $\mathbb{R}^{m+1}$.



Therefore we consider the following situation. In N we have $v = x_1$ and $u = (x_2, \ldots, x_m)$ and define the shadow $S_N = \{x_1 \,|\, C^N_{\max}(x_1, u) \leq t\}$. For N' we set $v = (x^1_1, x^2_1)$, $u = (x_2, \ldots, x_m)$ and $S_{N'} = \{(x^1_1, x^2_1) \,|\, C^N_{\max}(x^1_1, x^2_1, u) \leq t\}$. Let Q_1 be the distribution of X_1. Then using independence in N' we only need to show that

$$Q_1(S_N) \geq Q_1 \otimes Q_1(S_{N'})$$

holds.
Taking $m = 1$ we get $S_N = [0, a]$ and $S_{N'} = [0, b] \times [0, c]$. Without loss of generality we assume $b \geq c$.

Claim:
Under the assumptions above we get $a \geq c$

Assume not, then $a < c$. Then (c, u) gives

$$C^N_{\max}(a, u) > t \;\text{ in } N \quad \text{but} \quad C^{N'}_{\max}(c, c, u) \leq t \;\text{ in } N'.$$

but this contradicts

$$C^N_{\max}(c, u) = C^{N'}_{\max}(c, c, u)$$

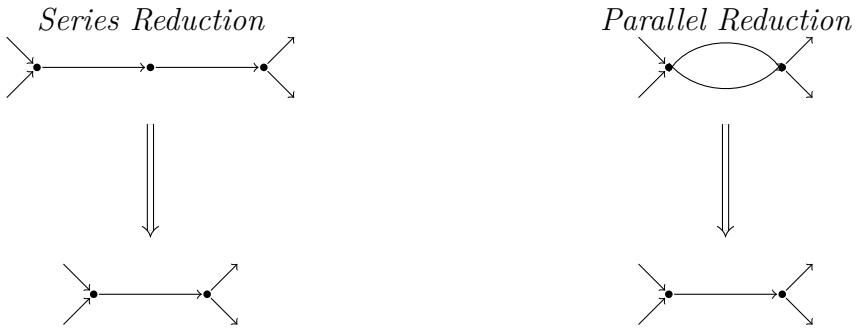
because all chains in N have the same length as in N'.
So $b \geq c$ which implies $a \geq c$. Now we can compute

$$Q_1(S_N) = Q_1([0,a]) \geq Q_1([0,c]) \geq Q_1([0,b]) \cdot Q_1([0,c]) = Q_1 \otimes Q_1(S_{N'}) \; .$$

Using the properties of the product measure and the independence of the distribution. □

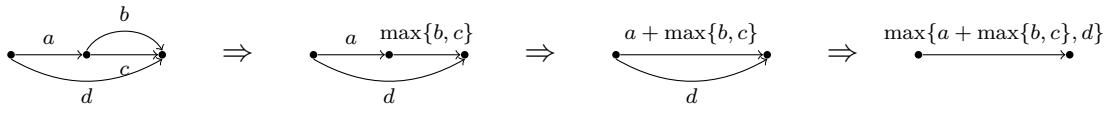
Our *sandwiching* may only be useful if the lower bound F_1 and he upper bound F_2 are easier to compute than the actual distribution function F. We can choose the networks N_1 and N_2 as special **series-parallel networks**.

Definition 17.4:
*A **series-parallel network** is a network that be reduced to* $(\{a,b\},(a,b)) = (V,A) = G$ *(so one job) by a sequence of series and parallel reductions.*[5]



Remark:
Series-Parallel networks can be recognized in linear time and a reduction sequence can be constructed in linear time. The makespan and makespan distribution can be calculated along a reduction sequence. △

Example 26 :
We look at deterministic processing times and get 'max' for parallel and '+' for series reductions:



♢

So we can compute $C_{\max}$ in the deterministic case by max and + for parallel and series reduction. So now we look at the stochastic counterparts to these operations

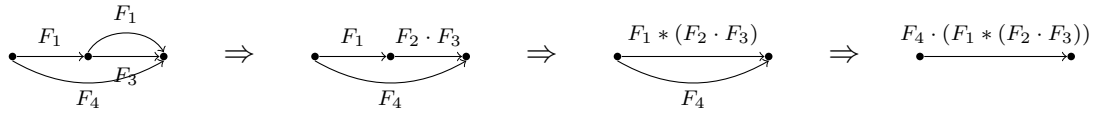
$$F_{\max\{X,Y\}} = F_X \cdot F_Y \qquad\qquad F_{X+Y} = F_X * F_Y$$

Where the convolution of distribution functions is defined by

$$F_1 * F_2(t) = \int_0^t F_1(t-x) f_2(x)\, dx \qquad f_2 \text{ density of } F_2$$

[5] These networks can be recognized in linear time.

Example 27 :
Our previous example with stochastic processing times.



$\diamondsuit$

Theorem 17.5:

(1) DF is NP-hard in the weak sense for series-parallel networks with 2-point processing time distributions.

(2) The reduction sequence algorithm computes for discrete distributions the makespan distribution function in a number of steps polynomial in

$$M = \max \textit{number of values of the makespan}$$

along the sequence.

Proof.
(1) Reduction from Partition
For an instance I of Partition, consisting of $a_1, \ldots, a_n, b \in \mathbb{N}$ with $\sum a_i = 2b$, we want to know whether there is a subset that adds up to exactly b. We construct an instance of 2 state DF-SP by constructing a network N with jobs 1 to n in a chain. Job j has the processing time distribution

$$X_j = \begin{cases} 0 & p = 0.5 \\ a_j & p = 0.5 \end{cases}$$

Assume there is a polynomial algorithm A that solves 2 state DF-SP. Then we use A to compute $F_N(t)$ for $t = b$ and $t = b - 1$. Then I is a Yes-instance if and only if there is a subset $J \subset \{1, \ldots, n\}$ such that $\sum_{j \in J} a_j = b$ if and only if $\{C^N_{\max} = b\}$ has positive probability if and only if F_N has a jump at $t = b$. This is equivalent to $F_N(b) > F_N(b-1)$. So we can decide in polynomial time if I is a Yes-instance of Partition.

(2) DF-SP can be solved in $O(M^2 n)$ time for discrete distributions, where M is the maximum number values of $C_{\max}$ along a reduction sequence and job processing times. This can be bounded from above by the sum of breakpoints per job over all jobs.
Assume that F_j (the distribution function of job j) is given by a list $t_1 < t_2 < \ldots < t_l$ of its jumps with values $F_j(t_1) \leq \ldots \leq F_j(t_l)$. Then $F_j(t)$ can be calculated by a scan through the list in $O(M)$ because $l \leq M$ at every stage along a reduction sequence. Now computing the product of two distribution functions takes $O(M)$ time and computing the convolution of two distribution functions takes $O(M^2)$ because

$$\mathbb{P}(X_i + X_j = t) = \mathbb{P}(\text{all combinations } X_i, X_j \text{ such that } X_i + X_j = t)$$

$\square$

Now we want t discuss specific bounds.

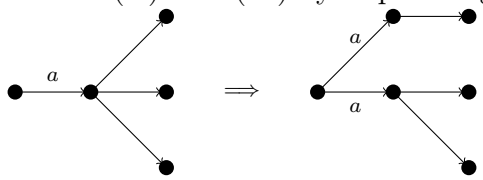
The bound of Dodin (1985)

The following algorithm gives an upper bound on the makespan distribution of a stochastic project network N.

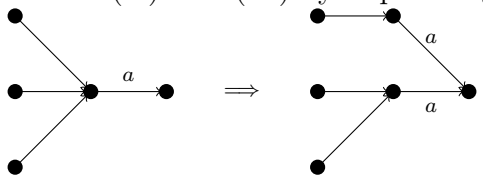
1: **while** $|A(N)| > 1$ **do**
2: **if** ser. red. possible **then** do series reduction
3: **else if** par. red. possible **then** do parallel reduction
4: **else** find (A) or (B) and duplicate the relevant arc (and get (A') or (B'))
5: **end if**
6: **end while**

The remaining single job is an upper bound for $F^N_{C_{\max}}$

Transform (A) into (A') by duplicating a



Transform (B) into (B') by duplicating a



Exercise

17.0 Show that Dodin's algorithm alway terminates with a single job.

Theorem 17.6:
Dodin's bound for a network N is exact if N is series-parallel. In general it is the distribution function of a series parallel network N' that contains N as a chain-minor.

Proof. Clear from the bounding principle. □

Bound of Spelde 1976

Now we want to use special series-parallel networks (parallel chains) to get upper and lower bounds. We consider disjoint chains for lower and all chains for upper bounds. This clearly works because of the minor property.
In addition we can approximate a distribution-free evaluation of Spelde's bound. For large networks we may assume that the chain length is normally distributed. We only need μ_j the mean value and σ_j^2 the variance of job j and get

$$\mu = \sum \mu_j \qquad \sigma^2 = \sum \sigma_j^2$$

But we may still have an exponential number of chains. We want to restrict to relevant chains. We compute the $1st$, $2nd$, ..., $k-th$ longest chains with respect to μ_j until $\mathbb{P}\{Y_k \geq Y_1\} \leq \varepsilon$ for a specified ε. Hereby Y is the length of the chain as

normal distribution. Then $F := F_1 \cdot \ldots \cdot F_k$ is an upper bound for $F^N_{C_{\max}}$.
To compute the k-longest paths one may use a k-shortest paths algorithm since N is acyclic. Such algorithms exist and run in $O(kn(m + n \log n))$ time.[6] This gives excellent and fast bounds in practice. A special case is PERT which considers only Y_1.
The *distance* of a partial order to a series-parallel order determines the quality of the bounds. Appropriate distance measures have been explored in combinatorics. We can measure the number of node reductions ($\rightarrow$ measure M_1) or the number of duplicated subexpressions ($\rightarrow$ measure M_2). For these measures the following results hold.

Theorem 17.7:
$M_1 \geq M_2$ and M_1 can be determined in polynomial time has been proven in 1992 by Bein, Kamburowski & Stallman. In 1994 Naumann proved $M_1 = M_2$

Exercises

17.1 Consider series-parallel partial orders and their comparability graphs. An undirected graph is called a complement-reducible graph or simply cograph if every induced subgraph H of G has the property that H or its complementary graph $\bar{H}$ is disconnected. Show that G is a cograph if and only if it is the comparability graph of a series-parallel partial order.

17.2∗ Let G be a finite undirected graph. Show G is a cograph if and only if G does not contain P_4 as induced subgraph. P_4 is a chain consisting of 4 nodes and 3 edges.

17.3 Let G be a finite partial order. Show G is series-parallel if and only if G does not contain the following Hesse diagram as induced suborder 

[6] See www.ics.uci.edu/~eppstein/bibs/kpath.bib

18 Bounds for Dependent Processing Times and the Makespan

The dependencies of the processing times are difficult to specify. Therefore we take a worst case approach for stochastic dependencies by [MN79] and [KH86].
Consider the expected tardiness

$$\mathbb{E}_Q[(C_{\max} - t)^+] = \mathbb{E}_Q[\max\{0, C_{max} - t\}]$$

of the makespan $C_{\max}$ in the worst case, i.e.,

$$\Psi(t) := \sup_Q \mathbb{E}_Q[(C_{\max} - t)^+]$$

Where the supremum is taken over all joint distributions with the given job processing time distributions as marginals. The function $\mathbb{E}_Q[(X - t)^+]$ is piecewise linear and convex for discrete random variables X.

Definition 18.1:
*A distribution P is called **stochastically smaller** than a distribution Q **in the convex sense** if for all monoton convex functions $f : \mathbb{R} \to \mathbb{R}$ holds*

$$P \leq_c Q \Leftrightarrow \int f\, dP \leq \int f\, dQ \Leftrightarrow \mathbb{E}[(X - t)^+] \leq \mathbb{E}[(Y - t)^+]$$

for all t if X, Y are real-valued random variables with distributions P, Q respectively.

Theorem 18.2: Meilijson & Nadas 1971
Let $\mathcal{P}$ be the class of joint distributions Q whose marginals Q_j (for job j) equal the processing time distributions. Then the following holds

(a) $\mathbb{E}_Q[(C_{\max} - t)^+] \leq \Psi(t)$ *for all* $Q \in \mathcal{P}$.

(b) There is a random variable Z with $\Psi(t) = \mathbb{E}[(Z - t)^+]$.

(c) If P_z is the distribution of Z, then $Q_{C_{\max}} \leq_c P_z$ *for all* $Q \in \mathcal{P}$.

(d) If G is series-parallel, then $P_z = Q_{C_{\max}}$ *for some* $Q \in \mathcal{P}$.

(e) $\Psi(t)$ is a tight upper bound for $\mathbb{E}_Q[(C_{\max} - t)^+]$ in the sense that for every t there is a distribution $Q_t \in \mathcal{P}$ such that $\Psi(t) = \mathbb{E}_{Q_t}[(C_{\max} - t)^+]$.

Proof. We show only (a). Consider a chain C of N and processing time vector $x = (x_1, \ldots, x_n)$. Now we can calculate a couple of estimates.

$$\begin{aligned}\sum_{j \in C} X_j - t &= \sum_{j \in C} x_j - t + \sum_{j \in C} (X_j - x_j) \\ &\leq (C_{\max}(x) - t)^+ + \sum_{j=1}^{n} (X_j - x_j)^+\end{aligned}$$

for all C and all x. If we maximize over all C and take the positive part and the expectation we get

$$\begin{aligned} \max_C \sum_{j\in C} X_j - t = C_{\max}(X) - t & \\ \leq (C_{\max}(X) - t)^+ & \\ \Rightarrow \quad \mathbb{E}_Q[(C_{\max}(X) - t)^+] \leq (C_{\max}(x) - t)^+ + \sum_{j=1}^n \mathbb{E}_Q[(X_j - x_j)^+] & \end{aligned}$$

for all x and all Q. So we can take the infimum over all x and the supremum over all Q because $\sum_j \mathbb{E}_Q[(X_j - x_j)^+]$ is independent of Q.

$$\sup_Q \mathbb{E}_Q[(C_{\max}(X) - t)^+] \leq \inf_x \left\{ (C_{\max}(x) - t)^+ + \sum_{j=1}^n \mathbb{E}_Q[(X_j - x_j)^+] \right\}$$

□

Now we want to solve the convex optimization problem

$$\Psi(t) = \min_{(x_1,\ldots,x_n)} \sum_{j=1}^n \mathbb{E}[(X_j - x_j)^+] \quad \text{such that } C_{\max} \leq t$$

There is a piecewise linear, convex and decreasing function $f(x_j)$ for every job j in the objective. We can interprete $f(x_j)$ as cost for executing job j with processing time x_j. So we get the side constraints to find processing times x_j that minimize the total cost and do not exceed the deadline t on the makespan. This leads to time-cost tradeoff problems.

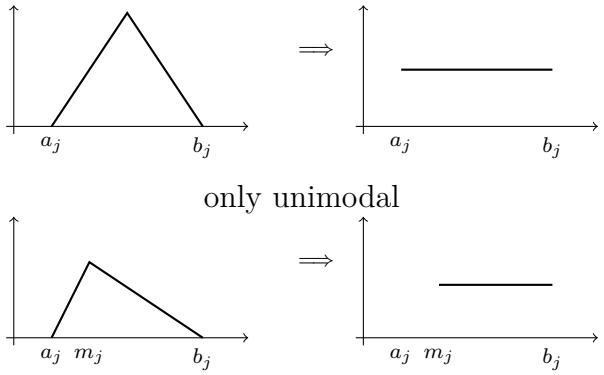
Time-Cost Tradeoff Problems

- classical network problem.
- is the dual of a min-cost-flow problem for fixed t (Fulkerson 1961)
- can be solved parametrically in t by a sequence of max-flow problems (Kelley 1961)
- very efficient in practice

Compatibility with Incomplete Information

If we have incomplete information about X_j we may simplify unimodal or unimodal and symmetric distributions to uniform distributions that are larger in the convex sense.

unimodal & symmetric



We still obtain upper worst case bounds if we use these uniform distributions when we only have incomplete information about the real distributions.

Exercise

18.1 $\Psi(t)$ for series-parallel networks

(a) Consider a 2-element (arcs X_1, X_2) chain and let F_1 and F_2 be the distribution functions of X_1, X_2. Show that the distribution of the 2-dimensional random variable $(F_1^{-1}(U), F_2^{-1}(U))$ with U uniformly distributed on $[0, 1]$, is the worst case distribution in the convex sense.

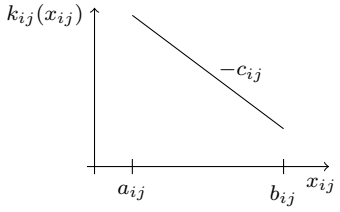
(b) Consider two jobs in parallel. With the notation of a) show that the distribution of $(F_1^{-1}(U), F_2^{-1}(1-U))$ is the worst case distribution in the convex sense.

(c) Use a) and b) to compute the worst case expected tardiness of a series parallel network.

19 Time-Cost Tradeoff Problems

Let us first consider the linear case. We are in the situation

- project network as arc diagram without parallel jobs
- for every job (i, j) an interval $I_{ij} = [a_{ij}, b_{ij}]$ of possible job durations
- for every job (i, j) a cost function k_{ij} with slope $-c_{ij}, c_{ij} > 0$



So $k_{ij}(x_{ij})$ denotes the cost of processing job (i, j) with processing time $x_{ij} \in I_{ij}$

- time limit t for the makespan

The goal is to execute the project at minimum cost within the given time limit.

$$\begin{aligned} \min & \sum_{(i,j)} k_{ij}(x_{ij}) =: k(x) \\ \text{s.t. } & C_{\max}(x) \leq t \\ & x = \text{ vector of chosen } x_{ij} \end{aligned}$$

Further we define the project cost curve $H(t)$ as the minimum cost for time t. This problem is called the (linear-) time-cost tradeoff problem. Note that we shorten jobs at a cost rate c_{ij} and want to find the right shortenings

$$k_{ij}(x_{ij}) = k_{ij}(b_{ij}) + (b_{ij} - x_{ij}) \cdot c_{ij}$$

The basic idea is to consider an optimal processing time vector x for t and characterize *optimal* tradeoffs to $t - \varepsilon$ for small ε in the arc diagram. We will show that we must shorten on a cut in the network of critical jobs.

Definition 19.1:
Let $D = (N, A)$ be the arc diagram of G, $N = \{1, \ldots, m\}$ where 1 denotes the source and m denotes the sink. A ***cut*** *$[S, T]$ of D is a partition of $N = S \dot\cup T$ of N with $1 \in S$ and $m \in T$. An arc (i, j) is critical if it is in a critical path for given x. Further $D_{crit} = (N_{crit}, E_{crit})$ denote the subnetwork of critical jobs.*

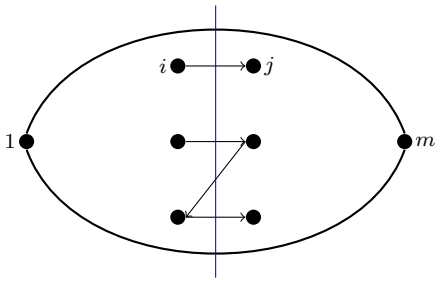
For a given x, let $\pi_i(x)$ denote the length of a longest path from 1 to i with respect to x. So an optimal x for t and z for $t - \varepsilon$ imply node potentials $\pi_i(x)$ and $\pi_i(z)$ respectively for every i. Thus

$$\pi_1(x) = \pi_1(z) = 0 \quad \pi_m(x) = t \quad \pi_m(z) = t - \varepsilon$$

holds and implies that

$$S := \{i \in N \mid \pi_i(x) = \pi_i(z)\} \quad T := N \backslash S$$

is a cut. The processing times have changed on some forward arcs in the cut and maybe elsewhere too. Let us consider what happens if we change processing times on a cut only.



Now we formulate two rules

(R1) If we shorten forward arcs by δ this reduces π_m by at least δ. But some $\pi_i(z)$ may be changed by a multiple of δ.

(R2) Shortening all forward arcs in a good cut of D_{crit} by δ reduces π_m by exactly δ.

Definition 19.2:
*A cut $[S,T]$ is **good** if every $s \in S$ can be reached from 1 by a directed path in S.*

Proof of (R2). Assume a topological sort of nodes. Let (i,j) be a forward arcs with largest i in S. Since $[S,T]$ is a good cut there is a path from 1 to i in S. Thus this path is only shortened by δ. Any path from j can not cross to S again because of the topological ordering. So if (i,j) is on a critical path then π_m reduces by exactly δ. □

But now backward arcs (k,i) in the cut have a slack δ. This can be exploited by lengthening backward arcs (if possible).

Lemma 19.3:
Let $[S,T]$ be a good cut. If we shorten all forward arcs and lengthen all backward arcs by δ then π_m is shortened by δ.

Proof. We already know that shortening on forward arcs decreases π_m by δ. Let $(i,j),(k,l)$ be forward arcs on a path with a backward arc between them. Then $-2\delta+\delta=\delta$, since every backward arc is contained between two forward arcs in that way. □

Definition 19.4:
*We call the following procedure **shortening on a (good) cut***

- *Shorten all forward arcs by δ and lengthen all backward arcs by δ (if their processing time permits).*

We define the cost rate of a (good) cut as

$$c[S,T] = \sum_{(i,j)f-a} c_{ij} - \sum_{\substack{(k,l)b-a \\ admissible}} C_{k,l}$$

How big can δ be ?

$$\delta = \min\{\delta_1, \delta_2, \delta_3\}$$

δ_1 $= \min\{\pi_j - \pi_i - x_{ij} \mid (i,j) \text{ not critical}\}$ meaning the amount of decrease until a non-critical job becomes critical.

δ_2 $= \min\{x_{ij} - a_{ij} \mid (i,j)$ forward arc in the cut$\}$ denoting the amount of decrease until a forward arc is at its minimum processing time

δ_3 $= \min\{b_{ij} - x_{ij} \mid$ backward arc and $x_{ij} < b_{ij}\}$ is the amount of increase until a backward arc that can be prolonged reaches its maximum processing time.

Necessary is of course that all forward arcs can be shortened.

Theorem 19.5:
Let x be optimal for $t > C_{\max}(a)$. Then there is a good cut $[S,T]$ in D_{crit} with associated $\delta > 0$ such that for every $\rho \in (0,\delta)$ the change of processing times according to (??) yields an optimal processing time vector y^ρ for $t - \rho$. So

$$y_{ij}^{\rho} = \begin{cases} x_{ij} - \rho & (i,j) \text{ is a forward arc of } [S,T] \\ x_{ij} + \rho & (i,j) \text{ is a backward arc of } [S,T] \text{ with } x_{ij} < b_{ij} \\ x_{ij} & \text{otherwise} \end{cases}$$

The total cost grows by the amount $\rho \cdot c[S,T]$

Proof. Later. □

Corollary 19.6:
The project cost curve $H(t)$ is piecewise linear and convex on

$$[t_{\min}, t_{max}] = [C_{\max}(a), C_{\max}(b)] \ .$$

It may be constructed as follows

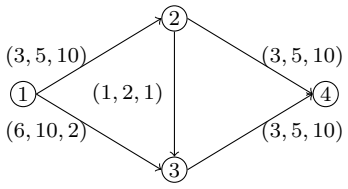
(1) Start at time $t = t_{\max}$. Then $x = (b_{ij})$ is optimal.

(2) Repeat until $t = t_{\min}$

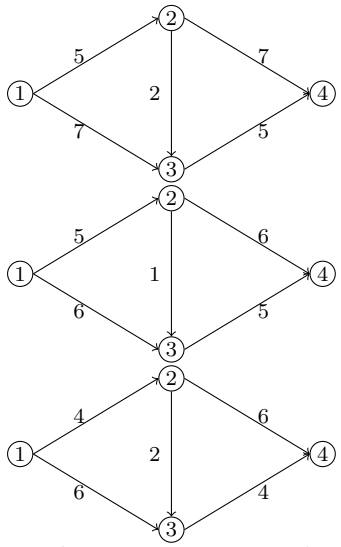
- *(a) Construct D_{crit} with respect to x.*
- *(b) Find a good cut with minimum cost rate in D_{crit} that can still be shortened.*
- *(c) Compute δ of this cut and change this processing times according to 19.4 into x^δ.*
- *(d) Set $t = t - \delta$ and $x = x^\delta$ the new optimal vector for $t - \delta$.*

Proof. Theorem 19.5 and integrality of a_{ij}, b_{ij} imply that $\delta \geq 1$. This proves termination. We will prove convexity later. □

Example 28 :



With (a_{ij}, b_{ij}, c_{ij}) and $t_{\max} = 15$, $t_{\min} = 9$. We start with all jobs at maximum duration. This gives $C_{\max} = 15$ and further $\delta = \delta_1 = 3 \leq 4 = \delta_2 < \delta_3 = \infty$ attained in $(2,3)$ and cost rate 4. This leads to



Looking at the graph and D_{crit} we obtain $C_{\max} = 12$, $\delta = \min\{\infty, 1, \infty\}$ and cost rate 6

Looking at the graph and D_{crit} we obtain $C_{\max} = 11$, $\delta = \min\{\infty, 2, 1\}$ and cost rate 18

Looking at the graph and D_{crit} we obtain $C_{\max} = 10$, $\delta = \min\{\infty, 1, \infty\}$ and cost rate 20

Now we reach $t_{\min} = 9$. Note that after shortening job $(2, 3)$ is no longer critical. $\diamondsuit$

Question: How to compute a good cut with minimum cost rate?
Idea: Use network flows and the max-flow-min-cut theorem. Meaning the maximum value of a $1 - m$ flow equals the minimum capacity of a $1 - m$ cut.
We call a flow feasible if it respects lower and upper capacities

$$l_{ij} \leq x_{ij} \leq u_{ij}$$

and flow conservation holds on $V \backslash \{1, m\}$. The value of the flow is given by

$$\sum_{j=1} x_{1,j} - \sum_{j=1} x_{j,1}$$

and the capacity of the cut $[S, T]$ is given by

$$\sum_{(i,j) \text{ forward}} u_{ij} - \sum_{(i,j) \text{ backward}} l_{ij} \ .$$

We define the cost rate of the cute $[S, T]$ as

$$\sum_{(i,j) \text{ forward}} c_{ij} - \sum_{(i,j) \text{ backward}} c_{ij}$$

where we only sum up over arcs that can be lengthened. Therefore we set in D_{crit}

$$u_{ij} := \begin{cases} c_i j & \text{if } x_{ij} > a_{ij} \\ \infty & \text{otherwise} \end{cases} \qquad l_{ij} := \begin{cases} c_{ij} & \text{if } x_{ij} < b_{ij} \\ 0 & \text{otherwise} \end{cases} \ .$$

From flow theory we know that any max flow algorithm producing good cuts will find a max flow and a good cut of minimum capacity. Here we find flow augmenting paths by breadth first search (BFS) which guarantees good cuts.

Theorem 19.7:

(1) Every good cut with minimum cost rate can be found by augmenting path algorithms for maximum flows in $O(n^3 \log n)$.

(2) The zero flow is feasible at $t_{\max}$. The current flow remains feasible when the capacities are changed. Thus H is convex.

(3) $H(t)$ can be calculated in $O(Mn^3 \log n)$ where M is the number of cut calculation which is bounded from below by the number of breakpoints of $H(t)$ (these can be exponentially many).

Proof.

(1) Follows from flow theory (especially Goldberg & Tarjan).

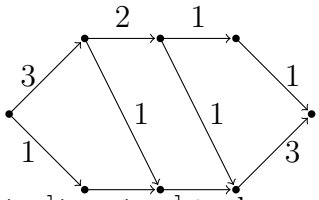
(2) Easy verification.

(3) Is obvious except for the exponential example.

□

The algorithm is based on the theorem 19.5.

Sketch of Proof of theorem 19.5 using examples.

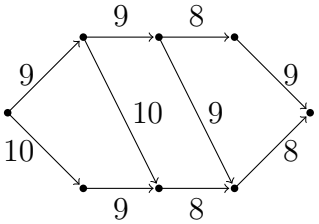


Every job has $a_{ij} = 1$ and $b_{ij} = 10$. The numbers on the edges are the c_{ij}.

Consider $t \in]t_{\min}, t_{\max}]$ and x optimal for t. The problem is that an optimal processing time vector for $t - \rho$ may be obtained by shorting several jobs in D_{crit} that are scattered all over the network.
In our example graph $x = (10, \ldots, 10)$ is optimal for $t = 40 = t_{\max}$ and every job is critical. Consider $\bar{\delta}$ defined as δ but over all arcs of D_{crit}.

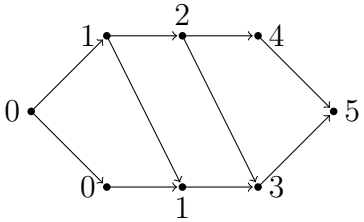
$$\left.\begin{array}{lcll} \bar{\delta}_1 & = & \infty & \text{all arcs are critical} \\ \bar{\delta}_2 & = & 9 & \text{all arcs can be shortened by 9} \\ \bar{\delta}_3 & = & \infty & \text{no arc can be lengthened} \end{array}\right\} \Rightarrow \bar{\delta} = 9$$

Let $\rho_0 \in (0, \bar{\delta}]$ and let y be optimal for $t - \rho_0$.



This is the y we work with. We also have $t - \rho = 35$.

Define for each $i \in V$ the potential difference $\delta\pi_i = \pi_i(x) - \pi_i(y)$. Let $\Delta\pi_1 < \ldots < \Delta\pi_l$ be the different $\delta\pi_i$ values of all i.



Our example with the potential differences $\delta\pi_i$. We get

$$\Delta\pi_1 = 0 < 1 < 2 < 3 < 4 < \Delta\pi_l = 5$$

(1) $[S_k, T_k]$ with $S_k := \{i \in V \mid \delta\pi_i \leq \Delta\pi_k\}$ and $T_k = V \backslash S_k$ is a good cut for all k.

(2) y can be obtained from x by the following algorithm
FOR $k := 1$ TO $l - 1$ DO
   (a) $\Delta_k := \Delta\pi_k - \Delta\pi_{k-1}$
   (b) $x_{ij} := x_{ij} - \Delta_k$ for all forward arcs in $[S_k, T_k]$
   (c) $x_{rs} := x_{rs} + \Delta_k$ for all backward arcs in $[S_k, T_k]$

(3) Let $[S_r, T_r]$ be the cut with smallest cost rate among the $[S_k, T_k]$. Let z be the processing time vector obtained by changing x on $[S_r, T_r]$ by ρ_0. Then z is optimal for $t - \rho_0$.

Proof. We can estimate the change of the total cost for $x \to y$ by

$$\begin{aligned}\sum_{k=1}^{l-1} c[S_k, T_k] \cdot \Delta_k &\geq \sum_{k=1}^{l-1} c[S_r, T_r] \cdot \Delta_k \\ &= c[S_r, T_r] \cdot \rho_0\end{aligned}$$

Which corresponds to the change from x to z. □

(4) The decrease on the cut $[S_r, T_r]$ is optimal for all $\rho \in (0, \bar{\delta}]$.
Let $\rho_1, \rho_2 \in (0, \bar{\delta}]$ with best cuts $[S_1, T_1]$ and $[S_2, T_2]$ according to (3). We may assume without loss of generality $\rho_1 \leq \rho_2$. We show that the cost rates of the two cuts are equal. If the cost rate for $[S_1, T_1]$ would be strictly less than the one for $[S_2, T_2]$ then decreasing on $[S_1, T_1]$ by $\min\{\rho_1, \rho_2\}$ would give a better solution for $t - \rho_1$ on the other cut. This gives a contradiction.

(5) So far we have decreased by at most $\bar{\delta}$. Now let us consider δ defined by the best cut $[S_r, T_r] =: [S, T]$. Then we have $\bar{\delta} \leq \delta$ and thus by (4) that $[S, T]$ is optimal for $t - \bar{\delta}$ and z. All cuts that can be decreased at time $t - \bar{\delta}$ and z can also be decreased at time t and x by definition of $\bar{\delta}$. Therefore $[S, T]$ is still the best at $t - \bar{\delta}$ and now ca be further decreased until δ.

□

Back to the computation of $\Psi(t) = \min\{\sum \mathbb{E}[(X_{ij} - x_{ij})^+\} \mid C_{\max}(x) \leq t\}$. Theorem **??** ensures that there is a random variable Z with $\Psi(t) = \mathbb{E}[(Z - t)^+]$ and that the first piece of $\Psi(t)$ will have the slope -1. So in the time cost trade off computation with cost functions $k_{ij}(x_{ij}) = \mathbb{E}[(X_{ij} - x_{ij})^+]$ we stop when the cost rate of the current minimum cut is greater than or equal to 1 and set the slop to the left of the current time to -1.

Exercise

19.1 Generalize the computation of $H(t)$ by flow methods to the case that all cost functions k_{ij} are piecewise linear and convex.

20 More on Project Scheduling with Resource Constraints

So far we have seen complexity results for machine scheduling problems. Project scheduling with resource constraints is much harder. These problems include a partial order for the jobs and a system of forbidden sets, where jobs may require several resource types. This problem is called resource constrained project scheduling problem (RCPSP). We will show the hardness of approximation and reoptimization for $C_{\max}$ and deterministic processing times.

Theorem 20.1:
RCPSP *is as hard to approximate as vertex coloring of graphs, i.e., unless* $P = NP$, *there is no approximation algorithm with a performance guarantee of* $n^{1-\varepsilon}$ *for every* $\varepsilon \in [0, 1)$, *where* n *is the number of jobs.*

Proof. Transform and instance of COLORING into an instance of RCPSP.
The COLORING consists of an undirected graph G a number $k \in \mathbb{N}$ and the question if there is a vertex coloring with at most k colors.
The RCPSP instance will be a graph $V = V(G)$ with $x_j = 1$ for all $j \in V$, no precedence constraints, a number $k \in \mathbb{N}$, $\mathcal{F} = E(G)$ and the question if there is a schedule with $C_{\max} \leq k$.
The time slots correspond to color classes. Thus the inapproximability result for coloring applies [Zuc07]. □

Complexity of Reoptimization

UPDATE-ADD (update after adding a forbidden set)
For a given instance I of RCPSP and an optimal schedule we want to find an optimal schedule for I' (I plus one new forbidden set).
Define **UPDATED-DELETE** analogously.

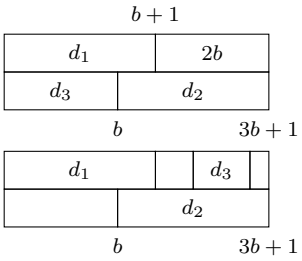
Theorem 20.2:
UPDATE-ADD is NP-hard in the weak sense.

Proof. Reduction from PARTITION[7]
For given numbers $a_1, \ldots, a_n \in \mathbb{N}$ with $\sum a_i = 2b\,, b \in \mathbb{N}$ we want to know if there is a subset $I \subset \{1, \ldots, n\}$ with $\sum_{i \in I} a_i = b$. From this instance of PARTITION we will construct an instance I of $RCPSP$ as follows:
We define

$$\begin{array}{rccc} \text{jobs} & \{1, \ldots, n\} & \cup & \{d_1, d_2, d_3\} \\ \text{processing times} & a_1, \ldots, a_n & & b+1, 2b+1, b \end{array}$$

with no precedence constraints and $\mathcal{F}$ consists of all 3-element sets and all sets of the form $\{d_3, j\}$ for $j = 1, \ldots, n$. So this corresponds to a 2-machine problem. Further we set $\kappa = C_{\max}$. Thus an optimal schedule looks like

[7] From ADM 1 we know that an NP-complete decision problem leads to an NP-hard optimization problem.



Now we add a new forbidden set $\{d_1, d_3\}$. If $C_{\max} = 3b+1$ then d_3 must be parallel to d_2. A possible schedule with $C_{\max} = 3b+1$ must look as follows

with d_3 and d_2 without loss of generality right shifted.

So deciding whether $C_{\max} \leq 3b + 1$ is as hard as deciding whether the instance of PARTITION is a Yes-instance. □

Theorem 20.3:
UPDATED-DELETE is NP-hard in the strong sense.

Proof. Reduction from COLORING PLANAR GRAPHS WITH MAXIMUM DEGREE $\Delta \leq 4$.[8]
For a given undirected planar graph G with degree $\Delta \leq 4$ we want to know if there is a vertex coloring of G with 3 colors. By the theorem of Brooks (theorem **??**) the problem to decide if a planar non-bipartite graph with $\Delta \leq 4$ can be colored with only 3 or 4 colors is NP-hard.
Reduction to RCPSP
So take G planar with $\Delta \leq 4$ etc, and define $\bar{G} = G + K_4$ with K_4 as a separate component. We transform $\bar{G}$ into an instacne of RCPSP as in theorem 20.4. We have no precedence constraints, $\kappa = C_{\max}$, $x_i = 1$ and $\mathcal{F} = E(\bar{G})$. Then an optimal schedule has $C_{\max} = 4$ because $\bar{G}$ contains a K_4. Assume that we know such an optimal schedule. We modify the RCPSP instance by deleting a forbidden set $\{u, v\}$ (an edge of K_4 in $\bar{G}$). Now we must decide if G can be colored with 3 or 4 colors. □

Theorem 20.4: Brooks (1941)
Every graph that is not K_n and not an odd cycle (C_{2k+1}) can be colored with Δ (maximum degree) colors.

[8] This is strongly NP-hard (see Garey et al. 1979)

21 Generalized Project Scheduling Problems and IP-Models

We divide this topic into three paragraphs:

21.1 Generalize Precedence Constraints by Time Lags.

21.2 Time Lags and Start Dependent Costs. (Uses flow methods)

21.3 Time Lags and Resource Constraints. (Uses Lagrange Relaxations)

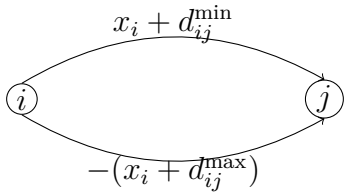
21.1 Generalize Precedence Constraints by Time Lags

$$\begin{array}{rlll} \text{precedence constraints} & i < j \Leftrightarrow & S_i + x_i \leq S_j & \\ \text{time lags} & & S_i + d_{ij} \leq S_j & d_{ij} \in \mathbb{R} \end{array}$$

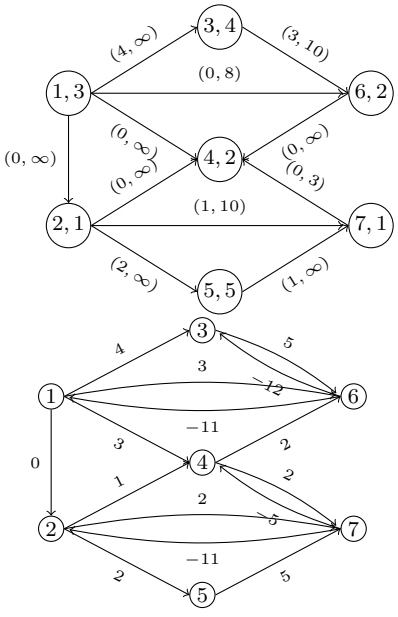
Interpretation of negative d_{ij}

$$S_i + d_{ij} \leq S_j \Leftrightarrow S_i \leq S_j - d_{ij}$$

So $S_j - d_{ij}$ is a deadline for the start of i relative to the start of j.
Time lags express lower and upper bounds on the start times and also the completion times of jobs relative to other jobs.
Time lags are often referred to as minimum (lower bounds, $d_{ij} \geq 0$) time lags and maximum (upper bounds, $d_{ij} \leq 0$) time lags. We will use the standard for $S_i + d_{ij} \leq S_j$ and represent it in a digraph by the arc



Example 29 :



Here is a larger example with different lags indicated by start and end of the arrows. On each arc we have $(d_{ij}^{\min}, d_{ij}^{\max})$ and in each node we have the job number i and its processing time x_i.

This is the resulting digraph D.

♢

The new question is: Is there a feasible schedule $S = (S_1, \ldots, S_k)$ respecting the given time lags? It is clear that there is no feasible schedule if D contains a cycle of positive length with respect to the d_{ij}. This is also sufficient.

Theorem 21.1:

(a) A project network with time lags has a feasible schedule if and only if the associated digraph has no cycle of positive length.

(b) In that case, the earliest start of job j is given by the length of the longest path from an artificial start node s (with zero arcs to all other nodes) to j.

(c) Checking for the existence of a positive cycle and computing the earliest start vector can be done in $O(n^3)$ with the Bellman-Ford algorithm.

Proof. See ADM 1 for the existence of negative cycles, feasible potentials and conservative arc weights. □

21.2 Scheduling with Time Lags and Start Dependent Costs

We add two artificial nodes (0 for the project start and $n+1$ for the project end) with processing time 0 and get $V = \{0, 1, \ldots, n, n+1\}$. For now consider fixed deterministic processing times $p_j \in \mathbb{N} \cup \{0\}$ and a set $L \subset V \times V$ of normalized time lags. Assume that they admit a feasible schedule. Let w_{jt} denote the cost if job j is started at time t for $t = 0, 1, 2, \ldots, T$ where T is an upper bound on the makespan. Our objective is to find a schedule that respects the time lags and minimizes the total start time costs.
We consider time lags $S_i + d_{ij} \leq S_j$ and costs w_{jt} for starting job j at time t. We note that $p_0 = p_n = 0$ and that time lags with 0 or $n+1$ can be arbitrary. We want to minimize the sum of all starting time costs over all feasible schedules (assuming there is one). Let us now formulate this as an IP.

$$\min \sum_{j=0}^{n+1} \sum_{t=0}^{T} w_{jt} x_{jt}$$

$$\text{s.t.} \quad \sum_{t=0}^{T} x_{jt} = 1 \qquad \forall j \tag{21.1}$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+d_{ij}-1} x_{js} \leq 1 \quad \forall (i,j) \in L\,,\ \forall t \tag{21.2}$$

$$x_{jt} \in \{0, 1\} \tag{21.3}$$

The constraint (21.1) ensures that every job is started exactly once and (21.2) models the time lags. So if i starts at a and j at b and $a + d_{ij} > b$ then (21.2) is violated for $t = a$.
Now we can compute in advance the earliest start times $e(j) \geq 0$ and latest start

times $l(j) < T - p_j$ for all j and set $x_{jt} = 0$ for $t \notin [e(j), l(j)]$. Because of (21.1) we can lift all w_{jt} by a constant M such that

$$\bar{w}_{jt} := w_{jt} + M \geq 0$$

This changes the objective only by the constant $(n+1)M$. So we may assume without loss of generality $w_{jt} \geq 0$.
We want to transform this into a minimum cut problem. Therefore we define $D = (N, A)$, the graph for the minimum cut problem.

Nodes We add one node for every possible start time of every job and $l(j) + 1$ as well as an artificial source a and sink b. This gives

$$N = \{w_{jt} \,|\, j \in V\,,\, t = e(j), e(j)+1, \ldots, l(j)+1\} \cup \{a, b\}$$

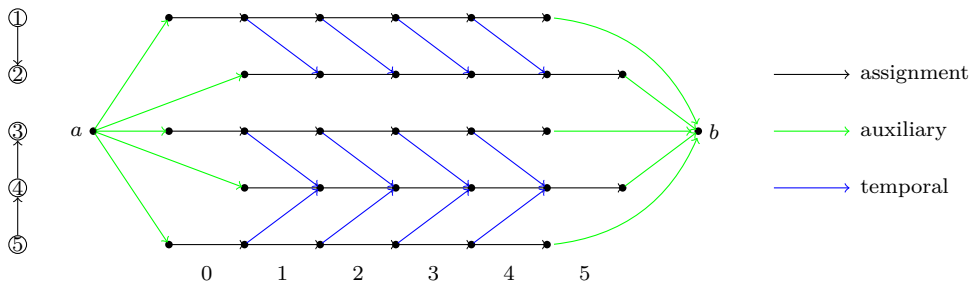
Arcs The graph shall have assignment arcs, temporal arcs and auxiliary arcs.

assignment $(v_{jt}, v_{j,t+1})$ for all $j \in V$

auxiliary $(a, v_{j,e(j)})\,,\, (v_{j,l(j)+1}, b)$ for all $j \in V$

temporal $(v_{it}, v_{j,t+d_{ij}})$ for all time lags $(i,j) \in L$ and for all t with $e(i)+1 \leq t \leq l(i)$ and $e(j)+1 \leq t + d_{ij} \leq l(j)$.

Example 30 :
We consider the following graph and its transformation. We have all $p_j = 1$, all $d_{ij} = 1$ and $T = 6$.



♢

Now the idea is to use assignment arcs to indicate the start of a job. We define the upper arc capacities $c(v_{jt}, v_{j,t+1}) = w_{jt}$ on assignment arcs and $c(\cdot) = \infty$ else as well as the lower capacities as 0. For our result, we are interested in special cuts, so called n-cuts.

Definition 21.2:
*An $a-b$ cut of D is an **n-cut** if the forward arcs of the cut contain exactly one assignment arc for every job.*

Lemma 21.3:
Let $(X, N\backslash X)$ be a minimum $a-b$-cut and $cap(X, N\backslash X) < \infty$. Then there is an n-cut with the same value. It can be computed from $(X, N\backslash X)$ in $O(n \cdot T)$ time.

Proof.

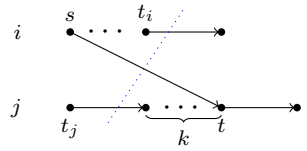
(1) The cut $(X, N\backslash X)$ contains at least one assignment arc for every jobs. As usual for job j we consider the j-path $a - v_{j,e(t)} - \ldots - v_{j,l(t)} - b$ with all the assignment arcs of j in the middle. Then this path must have a forward arc in $(X, N\backslash X)$.

(2) If $(X, N\backslash X)$ contains more than one assignment arc for some jobs, then we can construct an n-cut from it with the same capacity.
For such jobs let t_j be the first index such that $(v_{jt}, v_{j,t+1})$ is a forward arc in $(X, N\backslash X)$. We define the set

$$X^* := \{a\} \cup \bigcup_j \{v_{jt} \,|\, t \leq t_j\} \ .$$

Claim:
The cut $(X^*, N\backslash X^*)$ does not contain a temporal arc as forward arc.

Proof. Suppose it does contain a temporal arc (v_{is}, v_{jt}) as forward arc. So we have a time lag $d_{ij} = t - s$.



If t_j is a possible start time then $e(j) + 1 \leq t_j - k$. Furthermore we have $s - k \leq l(i)$ because $s - k \leq s \leq t_i \leq l(i)$ and we have $s - k \geq e(i) + 1$. To see that assume that the ladder inequality does not hold then we have the implications

$$\begin{aligned} & s - k < e(i) + 1 \\ \Rightarrow\ & t - k = s - k + d_{ij} < e(i) + d_{ij} + 1 \leq e(j) + 1 \\ \Rightarrow\ & t - k < e(j) + 1 \end{aligned}$$

This gives a contradiction.
Now the gathered inequalities allow us to conclude that $v_{i,s-k}$ is a start node for i and $v_{i,s-k} \in X^*$ and thus $v_{j,t-k} = v_{j,t_j+1} \in N\backslash X$. So the temporal arc $(v_{i,s-k}, v_{j,t+1})$ is a forward arc of $(X, N\backslash X)$ but this implies the contradiction $cap(X, N\backslash X) = \infty$. □

Claim:
The cut $(X^*, N\backslash X^*)$ has the same capacity as $(X, N\backslash X)$.

This is clear because $(X^*, N\backslash X^*)$ contains fewer assignment arcs, no temporal arcs and all w_{jt} are non-negative.

(3) The n-cut $(X^*, N\backslash X^*)$ can be constructed polynomial time from $(X, N\backslash X)$ in $O(nT)$. This is clear from the definition of X^*.

□

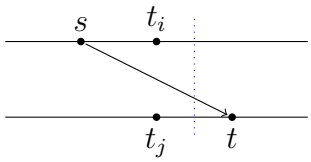
Theorem 21.4:
There is a one-to-one correspondence between n-cuts of D with finite capacities and feasible solutions of our scheduling problem defined by the IP above using the constraints (21.1) to (21.3), namely

$$x_{jt} = \begin{cases} 1 & \text{if } (v_{jt}, v_{j,t+1}) \text{ is in the cut } (X, N\backslash X) \text{ of } D \\ 0 & \text{otherwise} \end{cases}$$

The capacity $cap(X, N\backslash X)$ is equal to the value of the scheduling problem.

Proof.
(1) Let x be a feasible solution of the IP. Then x defines an n-cut because exactly one $x_{jt_j} = 1$. Set $X = \cup_j \{v_{jt} \mid t \leq t_j\} \cup \{a\}$. This defines a cut with capacities $cap(X, N\backslash X) = w(x)$ (Sum of assignment arcs). If there is another forward arc then that must be a temporal arc.

$v_{is} \to v_{jt}$ corresponds to a time lag $d_{ij} = t - s$



This implies $t_j - t_i < t - s = d_{ij}$, a contradiction to feasibility of x. Thus $w(x) = cap(X, N\backslash X)$. This implies that the optimal value of a minimum cut is less than or equal to the optimal value of the IP.

(2) Consider an n-cut $(X, N\backslash X)$ with finite value. Then the forward arcs in the cut define the start times t_j for j with $x_{jt_j} = 1$ and 0 else. Thus no temporal conditions are violated and we have that $cap(X, N\backslash X) = w(x)$. Therefore the minimum value of a cut is greater than or equal to the optimal value of the IP.

□

Corollary 21.5:
The scheduling problem with start-time dependent cost and time lags can be solved by computing a maximum $a-b$-flow and a corresponding $a-b$-cut in the digraph D.

Corollary 21.6:
This scheduling problem can be solved by computing a minimum $a-b$-cut in D or equivalently computing a maximum flow.

21.3 Lower Bounds for the RCPSP

We know that these problems including time lags, resource constraints and makespan objective are very hard. Many solution methods are based on IPs. Therefore we want a good IP formulation for RCPSP.
We introduce time indexed binary variables

$$x_{jt} := \begin{cases} 1 & \text{job } j \text{ starts at time } t \\ 0 & \text{otherwise} \end{cases}$$

We still want to minimize the makespan. So we only consider job n (project end) in the objective function.

$$\min \sum_t t \cdot x_{nt}$$

$$\text{s.t.} \sum_t x_{jt} = 1 \qquad \forall j \qquad \text{(IP:1.1)}$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+d_{ij}-1} x_{js} \leq 1 \qquad \text{(IP:1.2)}$$

$$\sum_j r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) \leq R_k \quad \forall k\,, \forall t \qquad \text{(IP:2)}$$

Where the last constraints (IP:2) model the resource constraints. We have R_k units available of resources k and job j needs r_{jk} units of type k. Now the constraints (IP:1.1) and (IP:1.2) would only give a pseudopolynomial problem but the constraints (IP:2) make the problem hard. So we want to simplify by relaxing the resource constraints.

Lagrange-Relaxation

We consider a general IP of the following form

$$\begin{aligned} &\min c^T x \\ &\text{s.t.} \quad Ax \geq b \quad \text{(hard)} && \text{(IP:H)} \\ &\qquad Bx \geq d \quad \text{(easy)} && \text{(IP:E)} \\ &\qquad x \text{ integer} \end{aligned}$$

We want to relax (IP:H) in terms of feasibility but punish its violation in the objective. This leads to the so called Lagrange Relaxation for $\lambda \geq 0$ fixed.

$$\begin{aligned} &\min c^T x + \lambda^T (b - Ax) =: L(x, \lambda) \\ &\text{s.t.} \quad Bx \geq d \\ &\qquad x \text{ integer} \end{aligned}$$

Now for all $\lambda \geq 0$ obviously holds

$$OPT(IP) \geq OPT(LR_\lambda)$$

We can try to improve these lower bounds by optimizing over λ.

$$\max_{\lambda \geq 0} \min_{x \text{ int}} L(x, \lambda) = \max_{\lambda \geq 0} \min_i \{c^T x^i + \lambda^T (b - Ax^i)\}$$

Therefore an optimal λ can be found by subgradient optimization. Hence we can compute the best (or a good) λ for $OPT(LR_\lambda)$. How good is the beast lower bound? The value $OPT(LR_\lambda)$ can be compared to the LP relaxation of the IP. Then

$$OPT(LR_{\lambda^{opt}}) \geq OPT(LP_{\text{relax}})$$

Equality holds if the polyhedron defined by the relaxed conditions $Bx \geq d$ is integral. We now apply the Lagrangian relaxation to the IP and relax the inequalities (IP:2). Now the objective becomes

$$\sum_t t \cdot x_{nt} + \sum_j \sum_t \left(\sum_{k \in R} r_{jk} \sum_{s=t}^{t+p_i-1} \lambda_{sk} \right) x_{jt} - \sum_t \sum_{k \in R} \lambda_{tk} R_k$$

We introduce weights

$$w_{jt} := \begin{cases} \sum_{k \in R} r_k \sum_{s=t}^{t+p_j-1} \lambda_{sk} & \text{if } j \neq n \\ t & \text{if } j = n \end{cases}$$

Now the objective becomes

$$\sum_j \sum_t w_{jt} x_{jt} - \sum_t \sum_{k \in R} \lambda_{tk} R_k$$

Where the second term is constant and therefore can be neglected. The resulting simplified problem is a scheduling problem with start-dependent cost.

$$\begin{aligned} \min & \sum_j \sum_t w_{jt} \cdot x_{jt} && \text{makespan + resource violation} \\ & \sum_t x_{jt} = 1 && \text{start every job} \\ & \sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+d_{ij}-1} \leq 1 && \text{respect temporal distances} \\ & x_j \in \{0,1\} \end{aligned}$$

The relaxed problem can be solved by max-flow computations. This gives a time-feasible solution, but not necessarily resource-feasible.
With better technology one may ask for more complex approaches and algorithm to achieve better results. In 2012 Jens Schulz showed in his Ph.D. thesis that this is possible by combining

- integer programming
- constraint programming
- SAT-solvers

Remark:
As an overall reference for scheduling theory we may refer to [Pin12] △

References

[Hag88] Hagstrom, J.N.: Computational Complexity of PERT Problems. In: *Networks* Vol.18 (1988), S. 139–147

[HSDW97] Hall, L. ; Schulz, A. ; D., Shmoys ; Wein, J.: Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Allgorithms. In: *Mathematics of Operatios Research* Vol.22 (1997), S. 513–544

[KH86] Klein-Haneveld, W.K.: Duality in Stochastic Linear and Dynamic Programming. In: *Lecture Notes in economics and Mathematical Systems* Vol.274 (1986), S. VIII – 295

[KL84] Korte, B. ; Lovasz, L.: Greedoids and Linear Objective Functions. In: *SIAM Journal on Algebraic and Discrete Methods* Vol.5 (1984), S. 229–238

[MN79] Meilijson, I. ; Nadas, A.: Convex Majorization with an Application to the Length of Critical Paths. In: *Journal of Applied Probability* Vol.16 (1979), S. 671–677

[MRW85] Möhring, R.H. ; Radermacher, F.J. ; Weiss, G.: Stochastic Scheduling II - Set Strategies. In: *Zeitschrift für Operations Research* Vol.29 (1985), S. 65–104

[MUV06] Megow, N. ; Uetz, M. ; Vredeveld, T.: Models and Algorithms for Stochastic Online Scheduling. In: *Mathematics of Operatios Research* Vol.31 (2006), S. 513–525

[Pin12] Pinedo, M.L.: *Scheduling*. Springer Verlag, 2012

[Zuc07] Zuckerman, D.: Linear Degree Extracors and the Inapproxability of Max Clique and Chormatic Number. In: *Theory Comput.* Vol. 3 (2007), S. 103–128

Index