A single machine on-time-in-full scheduling problem [*]

Marco Casazza[1], Alberto Ceselli[1], and Giovanni Righini[1]

[1]Università degli Studi di Milano, Dipartimento di Informatica, email:{firstname.lastname}@unimi.it

A relevant feature in many production contexts is flexibility. This becomes a key issue, for instance, in the case of third-party cosmetics manufacturing [1]. There, the core business is the production of high quality, fully custom orders in limited batches. Competition is pushing companies to aggressive commercial policies, involving tight delivery dates. At the same time, the custom nature of the orders makes it impossible to keep materials in stock; lead times are always uncertain, often making release dates tight as well, and ultimately yielding unexpected peaks of production loads.

At a scheduling stage, such an on-time-in-full (never split a job, always satisfy the customer within its delivery date with a single batch) company policy produces problems which are not only hard to solve by human experts, but often infeasible. As a consequence, delivery dates are sistematically violated, thereby lowering the perceived quality of service and triggering a loop of more aggressive commercial policies.

We consider a minimal relaxation of such a policy: in case scheduling all batches is infeasible, we leave the option of splitting some of them in two fragments (at a price), postponing the delivery date of the second fragment.

In this paper we focus on the combinatorial investigation of the fundamental case in which a single machine is available, with the target of using our findings in a column generation based algorithm for the general multi-machine multi-time-slot case. We formalize our main modeling assumptions, observe a few fundamental properties, and introduce an exact dynamic programming algorithm.

1 Models and assumptions

Formally, let J be a set of jobs (modeling customer batches). For each $j \in J$, let a release date r_j and a due date d_j be given. Let $\pi_j = p_j + q_j$ be the processing time of job j; when j is fragmented, we assume the processing time p_j (resp. q_j) of the first fragment (resp. second fragment) to be given.

In our setting, it is realistic to assume that scheduling is performed in fixed time slots (e.g. weekly), such that no job is left pending at the end of each slot (e.g. over the weekend). Let P be the total processing capacity of a particular machine in a particular time slot. From an application point of view, we expect a vast majority of the jobs to have $d_j - r_j > P$. However, we assume $\pi_j \leq P$.

Furthermore, let σ be the starting time of the optimization time slot. We preprocess release and due dates as $r_j = \max\{r_j - \sigma, 0\}$ and $d_j = \min\{d_j - \sigma, P\}$.

[*]Partially funded by Regione Lombardia, grant agreement n. E97F17000000009, Project AD-COM

Finally, we assume that no job preemption is possible, as it would imply additional setup time and storage costs.

Indeed, at a single machine stage, two conflicting objectives need to be considered: on one side, to perform as few splits as possible, as we expect good multi-machine multi-time-slot solutions to include few splits overall; on the other side, to schedule as many jobs as possible (potentially splitting them).

Our single machine on-time-in-full with fixed fragmentation scheduling problem (1-OTIFF) can be modeled as a bi-objective optimization problem as follows:

$$
\begin{aligned}
& \max \sum_{j \in J} x_j, \max \sum_{j \in J} z_j & & \quad (1)\\
\text{s.t. } & x_j \leq z_j & \forall j \in J & \quad (2)\\
& z_i + z_j \leq y_{ij} + y_{ji} + 1 & \forall i, j \in J : i \neq j & \quad (3)\\
& s_j + p_j z_j + q_j x_j \leq e_j & \forall j \in J & \quad (4)\\
& e_i \leq s_j + \mathcal{M}(1 - y_{ij}) & \forall i, j \in J, & \quad (5)\\
& x_j, z_j, y_{ij} \in \{0, 1\} & \forall i, j \in J & \quad (6)\\
& r_j \leq s_j, e_j \leq d_j & \forall j \in J & \quad (7)
\end{aligned}
$$

where variables x_j take value 1 if job j is performed in full, 0 otherwise, z_j take value 1 if job j is scheduled (either in full or after splitting), 0 otherwise, y_{ij} take value 1 if jobs i and j are both scheduled and i preceeds j, 0 otherwise, s_j (resp. e_j) take the starting time (resp. ending time) of job j (or are not influent if job j is not scheduled). Objective functions (1) maximize the number of scheduled jobs, and that of jobs which are scheduled in full. Constraints (2) ensure consistency between split and full job selection. Constraints (3) ensure consistency between z_j and y_{ij} values. Constraints (4) force consistency between job starting and ending time, when the job is selected. Constraints (5) are non-overlapping conditions. Constraints (6) and (7) define variable domains.

2 Properties and algorithms

It is not hard to prove the following:

Proposition 1. *When all due dates are identical, there always exists an optimal solution in which jobs are processed in order of non-decreasing release date*

and symmetrically

Proposition 2. *When all release dates are identical, there always exists an optimal solution in which jobs are processed in order of non-decreasing due date.*

Therefore, we partition the set of jobs J: those whose release date is the starting of the scheduling time slot (R), those not belonging to R whose due date is the ending of the scheduling time slot (D), and the remaining ones (S). We sort jobs by considering elements of R first, elements of D second and elements of S as last ones. We also fix an arbitrary order between elements in each subset, thereby fixing a total order between the jobs. We indicate $j > i$ (resp. $j < i$) whenever job j appears after (resp. before) job i in such a total ordering.

We remark that, as discussed in the modeling section, we expect S to be very small, being composed only by those jobs having both release and due date within the scheduling time slot.

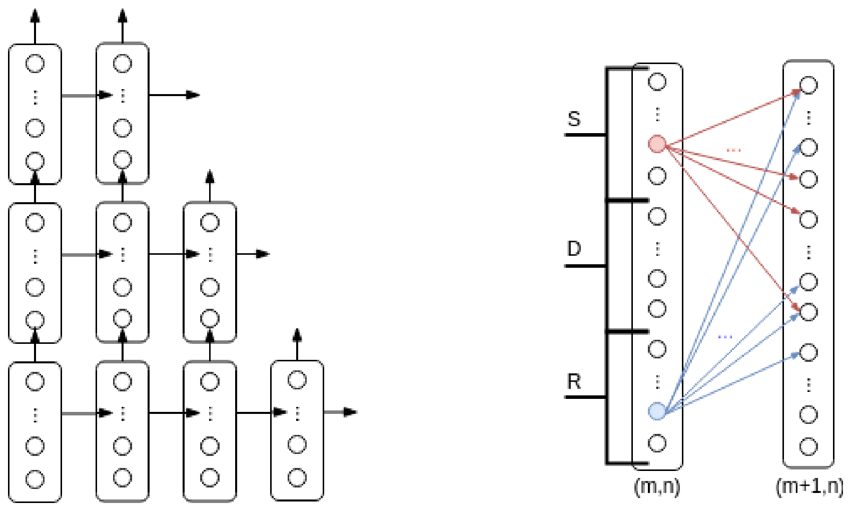

Figure 1: Layers grid structure (left): coordinates in the grid represent (m, n) values. Inter-layer arcs (right): blue arcs refer to a vertex i in either R or D, connected only to vertices $j > i$; red arcs refer to a vertex in S, connected to all vertices.

Second, we observe that finding suitable upper bounds on the maximum number f (resp. s) of jobs that are served in full (resp. partially), can be performed by simply solving a single-machine traditional scheduling problem, or even approximated by solving binary knapsack problems.

Then, we build a directed graph G having one layer for each $m = 0 \dots f$, for each $n = 0 \dots s$. Each layer (m,n) contains one vertex for each job; informally, we indicate a vertex to be in R (resp. D and S) if it is encoding a job in R (resp. D and S); therefore, each job is encoded by $f \cdot s$ vertices in G, one for each layer. Similarly, given two vertices i and j we indicate either $i < j$ or $i > j$ according to the relative position of the corresponding jobs in their total ordering. Each vertex i in S is connected to all vertices in both layer $(m+1,n)$ and layer $(m,n+1)$. Each vertex i in either R or D is instead connected only to vertices j having $j > i$ in both layers $(m+1,n)$ and $(m,n+1)$. Formally, we replace layer $(0,0)$ with a single dummy vertex 0.

The layers form a grid (Figure 1, left), arcs connect only adjecent layers (Figure 1, right).

1-OTIFF solutions are represented by paths, starting from 0, in such a graph. For finding optimal ones we design a dynamic programming algorithm, working as follows. At each vertex of G we consider *labels* encoding partial solutions, having the form (i, Q, t): t is the overall processing time spent so far, i is the last visited vertex *belonging to* $R \cup D$, Q is the set of jobs in S for which a corresponding vertex has already been visited in the partial solution. We initialize a single label of value $(0, \emptyset, 0)$ in the dummy vertex 0. Then we proceed in phases. At each phase l, we proceed by layers, considering in turn for each $v = 0 \dots l$ the layer $(l - v, v)$. For each layer we proceed by vertex: we perform extension operations, pushing its labels to neighbour vertices.

In particular, when extending a label (i, Q, t) in layer (m, n) from a vertex k to a vertex j

- if $j \notin S$, a new label $(j, Q, t + \pi_j)$ is created in layer $(m + 1, n)$ and a new label $(j, Q, t + p_j)$ is created in layer $(m, n + 1)$
- if $j \in S$, a new label $(i, Q \cup \{j\}, t + \pi_j)$ is created in layer $(m + 1, n)$ and a new label $(i, Q \cup \{j\}, t + p_j)$ is created in layer $(m, n + 1)$

We avoid extensions whenever $j \in Q$, in order to avoid including twice the same job in the partial solution, or $t > P$ in the new label, to avoid exceeding the available time slot. Furthermore we run dominance checks: taken two labels $\lambda' = (i', Q', t')$ and $\lambda'' = (i'', Q'', t'')$ belonging to the same vertex, we prune λ'' whenever $i' \leq i''$, $Q' \subseteq Q''$ and $t' \leq t''$, and at least one of these conditions is strict.

The algorithm terminates when no new label is created during a particular phase.

We can prove that

Proposition 3. *after termination, all Pareto-optimal solutions are encoded by labels*

and in particular, by labels in the top-right outermost border of non empty layers.

We discuss efficient implementations of our algorithm, and adaptations of speedup techniques from the literature.

Potentially, our algorithm has a high degree of computing parallelism, we therefore focus on effective techniques to enable concurrency.

As a perspective of future research, we remark that our algorithm naturally extends to the case in which prizes are assigned to the selection of jobs, in both full and split options. We expect such an extension to produce effective multiple pricing routines in a column generation setting for the multi-machine multi-time-slot version of the problem.

References

[1] Advanced Cosmetics Manufacturing, project website, https://ad-com.net/ (last access March 2019).