algorithms

MDPI

Article

Some Results on Shop Scheduling with S-Precedence Constraints among Job Tasks

Alessandro Agnetis [1,*], Fabrizio Rossi [2] and Stefano Smriglio [2]

[1] Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, SI 53100 Siena, Italy
[2] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università di L'Aquila, AQ 67100 L'Aquila, Italy; fabrizio.rossi@univaq.it (F.R.); stefano.smriglio@univaq.it (S.S.)
* Correspondence: agnetis@diism.unisi.it

Received: 9 October 2019; Accepted: 18 November 2019; Published: 25 November 2019

Abstract: We address some special cases of job shop and flow shop scheduling problems with s-precedence constraints. Unlike the classical setting, in which precedence constraints among the tasks of a job are finish–start, here the task of a job cannot start before the task preceding it has started. We give polynomial exact algorithms for the following problems: a two-machine job shop with two jobs when recirculation is allowed (i.e., jobs can visit the same machine many times), a two-machine flow shop, and an m-machine flow shop with two jobs. We also point out some special cases whose complexity status is open.

Keywords: job shop; flow shop; s-precedence constraints; exact algorithms; complexity

1. Introduction

This paper addresses a variant of classical shop scheduling models. While, in classical job shop or flow shop (as well as in the large majority of scheduling problems with precedence constraints), the task of a job cannot start before the previous task of the same job has *finished*, we address a situation in which each task of a job cannot start before the previous task of the same job has *started*. These types of constraints are known in the literature as *s-precedence constraints*. Scheduling problems with s-precedence constraints have been introduced by Kim and Posner [1] in the case of parallel machines. They showed that makespan minimization is NP-hard, and developed a heuristic procedure deriving tight worst-case bounds on the relative error. Kim, Sung, and Lee [2] performed a similar analysis when the objective was the minimization of total completion time of the tasks, while Kim [3] extended the analysis to uniform machines. Tamir [4] analyzed a parallel-machine problem in which traditional finish–start precedence constraints coexisted with s-precedence constraints (that he renamed *selfish precedence constraints*, giving an enjoyable dramatized motivation of the model), and established various worst-case bounds for classical dispatching rules which refer to specific structures of precedence constraints. Indeed, s-precedence constraints also arise in project management, called *start–start* precedence constraints (Demeulemeester and Herroelen [5]), as a result of the elaboration of a work breakdown structure (WBS) and of the coordination among different operational units. To our knowledge, none has addressed job shop and flow shop problems with s-precedence constraints so far.

The problem can be formally introduced as follows. We are given a set of n h $J^1, J^2, \ldots, J^n$, to be processed on a shop with m machines denoted as $M_1, \ldots, M_m$. Each job J^k consists of a totally ordered set of *tasks*, $J^k = \{O_1^k, O_2^k, \ldots, O_{n_k}^k\}$, $k = 1, \ldots, n$. Task O_i^k requires processing time p_i^k on a given machine $M(O_i^k)$, $i = 1, \ldots, n_k$. Tasks cannot be preempted. Task O_i^k can only start after task O_{i-1}^k is started; i.e., there is an s-precedence constraint between tasks O_{i-1}^k and O_i^k, for all $k = 1, \ldots, n$, $i = 2, \ldots, n_i$.

A *schedule* is an assignment of starting times to all tasks so that at any time each machine processes at most one task and all s-precedence constraints are satisfied. The problem is to find a feasible schedule that minimizes makespan.

We characterize the complexity of special cases of the problem, considering a fixed number of jobs and machines. Shop problems with few jobs occur when modeling synchronization and conflicts among processes share common resources. Examples of this situation include scheduling robot moves in flexible robotic cells (Agnetis et al [6]), aircraft scheduling during taxiing at an airport so that no aircraft collides (Avella et al. [7]), or, in container terminals, the synchronization of crane gantry movements once transportation tasks have been assigned (Briskorn and Angeloudis [8]).

The structure of the paper is as follows. In Section 2 we consider the job shop scenario, and give a polynomial time algorithm for the problem in which $n = 2$, $m = 2$, and each job can visit the machines several times (that is, *recirculation* [9] is allowed). In Section 3 we focus on the flow shop scenario. We show that the two-machine flow shop can be solved in linear time and we give a polynomial time algorithm for the m-machine problem with two jobs. In Section 4 we briefly discuss cases with $n > 2$ and point out open problems.

2. The Job Shop with Two Jobs and Two Machines

In this section we describe a polynomial algorithm for the job shop problem with two jobs and two machines; i.e., $J2|n = 2, s - prec|C_{\max}$. For notation simplicity, in this section we denote the two jobs as A and B, consisting of the sequence of *tasks*, $A = \{A_1, A_2, \ldots, A_{n_A}\}$ and $B = \{B_1, B_2, \ldots, B_{n_B}\}$, respectively. Task A_i (B_h) requires processing time p_i^A (p_h^B) on machine $M(A_i)$ ($M(B_h)$).

Obviously, if two consecutive tasks of the same job, say, A_i and A_{i+1}, require the same machine, then A_{i+1} has to wait for the completion of A_i, but if the machines required by the two operations are different, i.e., $M(A_{i+1}) \neq M(A_i)$, then A_{i+1} can start even if A_i has not completed yet. So, unlike the classical job shop setting in which precedence relations are finish-start, in our model it may actually happen that A_{i+1} even *completes* before A_i (the same of course applies to job B).

Given a partial schedule, the first unscheduled tasks of the two jobs will be referred to as the *available tasks*. Suppose now that one of the two machines, say M', has just completed a task, while the other machine, say M'', is still busy. If both the available tasks require M'', we say that machine M' is *blocked* and this certainly results in idle time on M'.

We let $A[i]$ and $B[h]$ denote the first i tasks of A and the first h tasks of B; i.e., $A[i] = \{A_1, A_2, \ldots, A_i\}$ and $B[h] = \{B_1, B_2, \ldots, B_h\}$.

Given A and B, consider any two task subsequences $X \subseteq A$ and $Y \subseteq B$. We want to characterize the schedules of $X \cup Y$ such that each task starts right after the previous task on the same machine has completed. More formally, a schedule of $X \cup Y$ is a *no-idle subschedule* (NIS) if, across the span of such a subschedule, the only machine idle time occurs, on one machine, after all the tasks of $X \cup Y$ have *started*. When $X = A[i]$ and $Y = B[h]$, for some $1 \leq i \leq n_A$ and $1 \leq h \leq n_B$, then we say that the NIS is an *initial no-idle subschedule* (INIS).

Consider Figure 1 and the task set $A[2] \cup B[2]$. The subschedule in Figure 1a is *not* an INIS for $A[2] \cup B[2]$, since on M_1 there is idle time before B_2 starts. On the contrary, in the case depicted in Figure 1b, the subschedule of $A[2] \cup B[2]$ is an INIS. Note that if we restrict our attention to the task set $A[2] \cup B[1]$, then the subschedule in Figure 1a is an INIS.

(**a**) (**b**)

Figure 1. In instance (**a**), the set $A[2] \cup B[2]$ does not form an INIS (initial no-idle subschedule); in instance (**b**) it does.

2.1. Generating Initial No-Idle Subschedules

We denote by $A[i, M_j]$ and $B[h, M_j]$, the subset of tasks of $A[i]$ and $B[h]$ respectively, requiring machine M_j, $j = 1, 2$; i.e.,

$$A[i, M_j] = \{A_r : r \leq i, M(A_r) = M_j\} \quad j = 1, 2,$$

$$B[h, M_j] = \{B_q : q \leq h, M(B_q) = M_j\} \quad j = 1, 2.$$

We also let $P(A[i, M_j])$ and $P(B[h, M_j])$ indicate the total processing time of tasks in $A[i, M_j]$ and $B[h, M_j]$; i.e.,

$$P(A[i, M_j]) = \sum_{r \in A[i, M_j]} p_r^A,$$

$$P(B[h, M_j]) = \sum_{q \in B[h, M_j]} p_q^B.$$

If an INIS of tasks $A[i] \cup B[h]$ exists, its makespan is given by

$$\max\{P(A[i, M_1]) + P(B[h, M_1]), P(A[i, M_2]) + P(B[h, M_2])\}.$$

Proposition 1. *In any optimal schedule, there are indices i and h such that the subschedule involving tasks $A[i] \cup B[h]$ is an INIS.*

In fact, given an optimal schedule, consider the subschedule of the tasks scheduled on the two machines from time 0 to the end of the first idle interval of the schedule, assuming, e.g., that such an idle interval occurs on M_1. If the subschedule is not an INIS, we can iteratively remove the last task scheduled on M_2 in the subschedule, until the definition of INIS is met.

In view of Proposition 1, we are only interested in schedules in which the initial part is an INIS. However, not all initial no-idle subschedules are candidates to be the initial part of an optimal schedule.

We first address the following question. Can we determine all operation pairs (i, h) such that an INIS of $A[i] \cup B[h]$ exists? We show next that this question can be answered in polynomial time.

The idea is to build the no-idle partial schedules from the beginning of the schedule onward. To this aim, let us define an unweighted graph G, which we call *initial no-idle graph*. Nodes of G are denoted as (i, h), representing a NIS of $A[i] \cup B[h]$ (for shortness, we use (i, h) also to denote the corresponding INIS). If the schedule obtained appending B_{h+1} to schedule (i, h) is still an INIS, we insert node $(i, h + 1)$ and an arc from (i, h) to $(i, h + 1)$ in G. Symmetrically, if the schedule obtained appending A_{i+1} to (i, h) is an INIS, we insert $(i + 1, h)$ and an arc from (i, h) to $(i + 1, h)$ in G.

As illustrated later on (cases $(i) - (iv)$ below), while building the graph G, we can also determine whether or not a certain INIS can be the initial part of an optimal schedule. If it can, we call it a *target node*.

Consider any node (i,h) in G, and the machine completing soonest in the INIS. Ties can be broken arbitrarily, but to fix ideas, suppose that M_2 is still busy when M_1 completes. (Note that, since there is no idle time, M_1 completes at time $P(A[i,M_1]) + P(B[h,M_1])$.) If $i < n_A$ and $h < n_B$, the two available tasks are A_{i+1} and B_{h+1}, and four cases can occur.

(*i*) $M(A_{i+1}) = M(B_{h+1}) = M_2$. In this case, M_1 is necessarily idle until M_2 completes (Figure 2a). Hence, there is no way to continue an INIS, and therefore node (i,h) has no outgoing arcs. In this case, (i,h) is a target node.

(*ii*) $M(A_{i+1}) = M_1$ and $M(B_{h+1}) = M_2$. In this case, when A_i completes, the only way to continue an INIS is to start task A_{i+1} on M_1 (Figure 2b). Thus we generate node $(i+1,h)$ and the arc from (i,h) to $(i+1,h)$, which is the only outgoing arc of (i,h). In this case as well, (i,h) is a target node.

(*iii*) $M(A_{i+1}) = M_2$ and $M(B_{h+1}) = M_1$. A symmetrical discussion to the previous case holds; i.e., the only way to continue an INIS is to start task B_{h+1} on M_1 (Figure 2c), so we generate node $(i,h+1)$ and the arc from (i,h) to $(i,h+1)$, which is the only outgoing arc of (i,h). In this case also, (i,h) is a target node.

(*iv*) $M(A_{i+1}) = M(B_{h+1}) = M_1$. In this case, the INIS can be continued in two possible ways; i.e., scheduling either A_{i+1} or B_{h+1} on M_1 (Figure 2b,c respectively). Therefore, (i,h) has two outgoing arcs, pointing towards nodes $(i+1,h)$ and $(i,h+1)$, respectively. However, in this case (i,h) is *not* a target node, since there is no point in keeping M_1 idle until the completion of M_2.

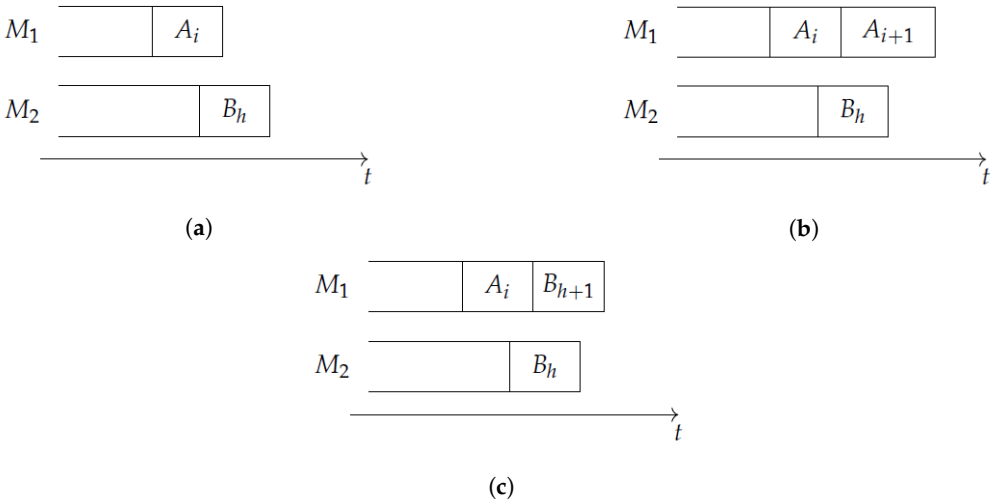

Figure 2. Possible scenarios when M_1 completes before M_2: (**a**) The INIS (i,h) cannot be continued, (**b**) it can only be continued scheduling A_{i+1}, and (**c**) it can only be continued scheduling B_{h+1}.

Clearly, if M_2 completes before M_1, in the four above cases the roles of M_1 and M_2 are exchanged. If either $i = n_A$ or $h = n_B$, the above cases simplify as follows, where we assume that $h = n_B$; i.e., job B is finished. (A symmetric discussion holds in $i = n_A$.)

(*v*) $M(A_{i+1}) = M_1$. In this case, we can continue an INIS starting task A_{i+1} on M_1. Thus we generate node $(i+1,h)$ and the arc from (i,h) to $(i+1,h)$, which is the only outgoing arc of (i,h). Node (i,h) is a target node.

(*vi*) $M(A_{i+1}) = M_2$. In this case, M_1 is necessarily idle until M_2 completes. Hence, there is no way to continue an INIS, and therefore node (i,h) has no outgoing arcs. In this case, (i,h) is a target node.

Again, the roles of the two machines are exchanged if M_2 frees up before M_1 in the partial schedule.

In conclusion, the question of whether a NIS exists for the task set $A[i] \cup B[h]$ is equivalent to asking whether node (i,h) can be reached from the dummy initial node $(0,0)$ on G.

A few words on complexity. Clearly, G has $O(n_A n_B)$ nodes, and each node has at most two outgoing arcs. The graph G can be built very efficiently. In fact, for each node (i,h), it can be checked in constant time, which condition holds among (i)–(iv) (or (v)–(vi) when one of the jobs is over), and hence whether or not it is a target node.

2.2. Minimizing the Makespan

Now we can address the main question. How to schedule the tasks on the two machines so that the overall makespan is minimized. The key idea here is that any active schedule can be seen as the juxtaposition of no-idle subschedules. In fact, suppose that after processing a certain task A_i, one machine stays idle until the other machine completes task B_h. It is important to observe that this may happen for one of *two* reasons:

- When a machine completes, it is blocked because both available tasks require the other machine;
- When a machine completes, there *is* one task the machine can process, but it might be profitable to wait for the other machine to free up another task.

Note that in both of these two cases (i,h) is a target node of G. On the contrary, if a machine completes a task while the other machine is still busy, and both available tasks require that machine (i.e., (i,h) is not a target node of G), with no loss of generality we can assume that the machine *will* immediately start one of them, since otherwise the schedule might turn out non-active (there is no point in waiting for the other machine to complete its task).

If t denotes the makespan of an INIS, the schedule after t is completely independent from the schedule before t. In other words, the optimal solution from t onward is the optimal solution of a problem in which t is indeed time 0, and the two jobs are $A \setminus A[i]$ and $B \setminus B[h]$. Hence, to address the overall problem, the idea is to build another, higher-level graph in which the arcs specify portions of the overall schedule.

Suppose that (i,h) is a target node of graph G, and consider the task sets $A \setminus A[i]$ and $B \setminus B[h]$. We can build a new *no-idle graph* on these sets, and call it $G(i,h)$. (Correspondingly, the graph previously denoted as G can be renamed $G(0,0)$.) Suppose that (r,q) is a target node in graph $G(i,h)$. This means that the tasks of the set $\{A_{i+1}, A_{i+2}, \ldots, A_r\} \cup \{B_{h+1}, B_{h+2}, \ldots, B_q\}$ form a NIS, that we denote by $[(i+1,h+1) \to (r,q)]$. It is convenient to extend the previous notation, letting $A[i+1,r,M_j]$ denote the set of tasks of $A[i+1,r]$ that require machine M_j, and analogously we let $B[h+1,q,M_j]$ be the set of tasks of $B[h+1,q]$ that require M_j. Their total processing times are denoted as $P(A[i+1,r,M_j])$ and $P(B[h+1,q,M_j])$. (The set previously denoted as $A[i,M_j]$ should now be written $A[0,i,M_j]$.)

We next introduce the (weighted) graph $\mathcal{G}$ as follows. As in G, nodes denote task pairs (i,h). There is an arc $[(i,h),(r,q)]$ if (r,q) is a target node in the graph $G(i,h)$; i.e., if the NIS $[(i+1,h+1) \to (r,q)]$ exists. Such an arc $[(i,h),(r,q)]$ is weighted by the length of the corresponding NIS; i.e.,

$$\max\{P(A[i+1,r,M_1]) + P(B[h+1,q,M_1]), P(A[i+1,r,M_2]) + P(B[h+1,q,M_2])\}.$$

Moreover, $\mathcal{G}$ contains a (dummy) initial node $(0,0)$ while the final node is (n_A,n_B). At this point the reader should have no difficulty in figuring out that the following theorem holds.

Theorem 1. *Given an instance of* $J2|n=2, s-prec|C_{\max}$*, the optimal schedule corresponds to the shortest path from* $(0,0)$ *to* (n_A,n_B) *on* $\mathcal{G}$*, and its weight gives the minimum makespan.*

Now, let us discuss complexity issues. The graph $\mathcal{G}$ can indeed be generated starting from $(0,0)$, and moving schedules forward. From each node (i,h) of $\mathcal{G}$, we can generate the corresponding no-idle

graph $G(i,h)$, and add to $\mathcal{G}$ all target nodes of $G(i,h)$. We then connect node (i,h) in $\mathcal{G}$ to each of these nodes, weighing the arc with the corresponding length of the NIS. If a target node was already present in $\mathcal{G}$, we only add the corresponding new arc. Complexity analysis is, therefore, quite simple. There are $O(n_A n_B)$ nodes in $\mathcal{G}$. Each of these nodes has a number of outgoing arcs, whose weight can be computed in $O(n_A n_B)$. Clearly, finding the shortest path on $\mathcal{G}$ is not the bottleneck step, and therefore, the following result holds.

Theorem 2. *$J2|n = 2, s - prec|C_{\max}$ can be solved in $O(n_A^2 n_B^2)$.*

Example 1. *Consider the following instance, in which job A has four tasks and job B two tasks.*

job	*1*	*2*	*3*	*4*
ine A	*5,M_1*	*1,M_1*	*4,M_2*	*6,M_1*
B	*4,M_2*	*7,M_2*	–	–

Figure 3a depicts the graph $G(0,0)$, in which all nodes are target nodes. Figure 3b shows the INIS $[(0,0) \to (2,2)]$. At the end of this INIS, machine M_1 is blocked, since the next task of A requires M_2 and job B is already finished. Notice that $[(0,0) \to (2,2)]$ is the longest INIS which can be built, but the optimal solution does not contain it. Figure 4a shows the best schedule which can be attained when the INIS $[(0,0) \to (2,2)]$, having makespan 17. Figure 4b shows the optimal schedule, having makespan 16. The optimal schedule consists of two no-idle subschedules; namely, the INIS $[(0,0) \to (1,1)]$ (containing tasks A_1 and B_1 and corresponding to arc $[(0,0),(1,1)]$ on $\mathcal{G}$), and the NIS $[(2,2) \to (4,2)]$ (containing tasks A_2, A_3, A_4 and B_2 and corresponding to arc $[(1,1),(4,2)]$ on $\mathcal{G}$). For illustrative purposes, Figure 5 shows the graph $G(1,1)$. Notice that in such a graph, (2,1) is not *a target node.*

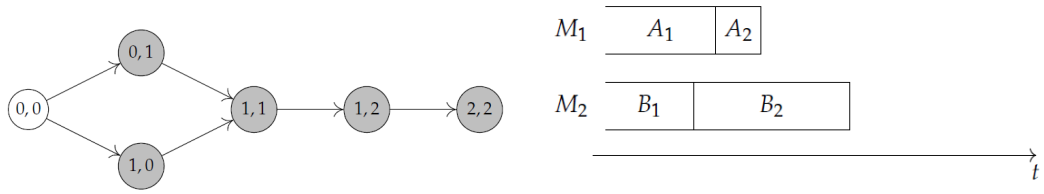

Figure 3. (**a**) The graph $G(0,0)$ in the example. (**b**) The INIS $[(0,0) \to (2,2)]$.

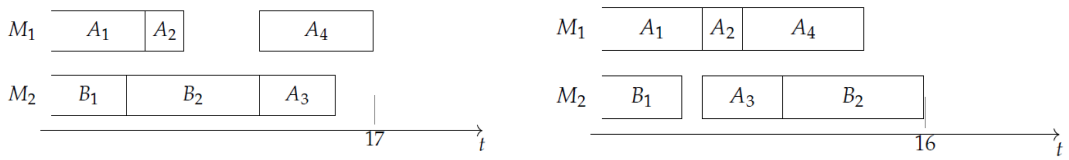

Figure 4. (**a**) The best schedule starting with the INIS $[(0,0) \to (2,2)]$, and (**b**) the optimal schedule in the example.

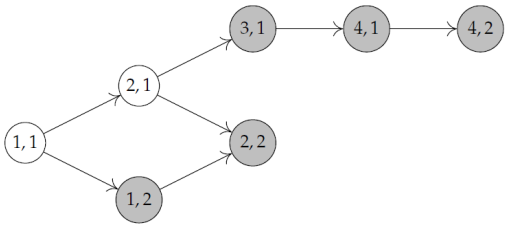

Figure 5. Graph $G(1,1)$ in the example.

3. Flow Shop

In this section we consider the flow shop problem, i.e., $F|s-prec|C_{\max}$, in which the job set J contains n jobs, and job J^k requires processing time p_j^k on machine M_j (here we use index j for both tasks and machines, as there is exactly one task per machine). While in the classical problem $F||C_{\max}$ a job cannot start on machine M_j before it is completed on M_{j-1}, in $Fm|s-prec|C_{\max}$, a job can start on machine M_j as soon as it is started on M_{j-1}.

3.1. Two-Machine Flow Shop ($F2|s-prec|C_{\max}$)

We next consider the two-machine flow shop problem, so p_1^k and p_2^k denote the processing times of job J_k on M_1 and M_2 respectively, $k = 1, \ldots, n$. Note that, as in the classical $F2||C_{\max}$, with no loss of generality we can assume that in any feasible schedule the machine M_1 processes all the jobs consecutively with no idle time between them. We next show that problem $F2|s-prec|C_{\max}$ can be solved in linear time.

Proposition 2. *Given an instance of $F2|s-prec|C_{\max}$, there always exists a schedule σ^* having makespan $\max\{\sum_{k=1}^n p_1^k, \sum_{k=1}^n p_2^k\}$, which is therefore optimal.*

Proof. Given an instance of $F2|s-prec|C_{\max}$, partition the jobs into two sets, J' and J'', such that $J' = \{J^k | p_1^k \leq p_2^k\}$ and $J'' = J \setminus J'$. Then, build σ^* by scheduling, on both machines, first all jobs of J' in arbitrary order, and then all jobs of J'', also in arbitrary order. If we let $C(1)$ and $C(2)$ denote the completion time of the last job of J' on M_1 and M_2 respectively, one has $C(1) < C(2)$. From the definition of J', one gets that up to $C(2)$, no idle time occurs on M_2. From then on, all jobs of J'' are scheduled, and two cases may occur. (i) No idle time ever occurs on M_2, in which case the makespan equals $\max\{\sum_{k=1}^n p_1^k, \sum_{k=1}^n p_2^k\}$. (ii) Some idle time occurs on M_2. Consider the first time that M_2 is idle and M_1 is still processing a job J_k. Upon completion of J_k, the two machines will simultaneously start the next job, say, $J_{\bar{k}}$, but since $J_{\bar{k}} \in J''$, M_1 will still be processing it while M_2 returns idle. Since all remaining jobs belong to J'', this will happen for each job until the end of the schedule. In particular, when the last job is scheduled, again, M_2 completes first, so in conclusion, the makespan of σ^* is $\sum_{k=1}^n p_1^k$. □

The above proof contains the solution algorithm. For each job J^k, put it into J' if $p_1^k \leq p_2^k$ and in J'' otherwise. Then, schedule all jobs of J' followed by all jobs of J'' (in any order). Since establishing whether a job belongs to J' or J'' can be done in constant time, and since jobs can be sequenced in arbitrary order within each set, we can conclude with the following result.

Theorem 3. *$F2|s-prec|C_{\max}$ can be solved in $O(n)$.*

While $F2|s-prec|C_{\max}$ appears even simpler than the classical $F2||C_{\max}$, one may wonder whether other simplifications occur for $m > 2$. While the complexity status of $Fm|s-prec|C_{\max}$ is open, we point out a difference between $Fm||C_{\max}$ and $Fm|s-prec|C_{\max}$, which may suggest that the problem with s-precedence constraints is not necessarily easier than the classical counterpart.

It is well known [10] that in $Fm||C_{\max}$ there always exists an optimal schedule in which the job sequences on M_1 and M_2 are identical, and the same holds for machines M_{m-1} and M_m. (As a consequence, for $m \leq 3$ the optimal schedule is a permutation schedule.) This is no more true in $Fm|s-prec|C_{\max}$, even with only two jobs.

Example 2. *Consider an instance with three machines and two jobs, A and B:*

j	1	2	3
ine A	4,M_1	6,M_2	1,M_3
B	10,M_1	4,M_2	9,M_3
ine			

Scheduling the jobs in the order AB on all three machines, one gets $C_{max} = 15$, and the makespan is attained on machine M_3 (see Figure 6a). If they are scheduled in the order BA on all three machines, $C_{max} = 16$, and in this case the value of the makespan is attained on M_2 (Figure 6b). If jobs are scheduled in the order AB on M_1 and BA on M_2 and M_3, then $C_{max} = 14$ (all three machines complete at the same time, Figure 6c), and this is the optimal schedule.

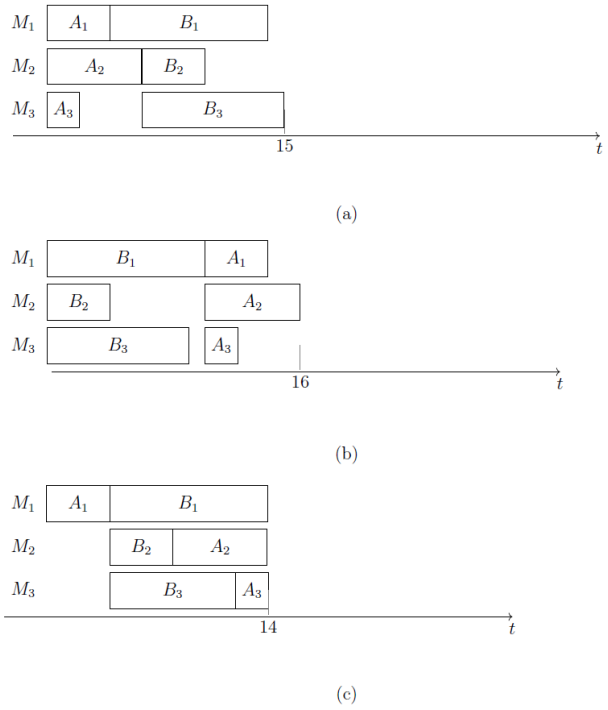

Figure 6. Three schedules for Example 2.

3.2. Flow Shop with Two Jobs and m Machines ($Fm|n = 2, s-prec|C_{\max}$)

In what follows we give an Algorithm 1 that solves $Fm|n = 2, s-prec|C_{max}$. Again we denote the two jobs with A and B, and by p_j^A and p_j^B the processing time of jobs A and B on machine M_j, $j = 1, \ldots, m$, respectively. Notice that a schedule is fully specified by the order of the two jobs on each machine, either AB or BA. In what follows, for a given schedule and a given machine, we call *leader* the job scheduled first and *follower* the other job. So if, on a given machine, the jobs are sequenced in the order AB, then, on that machine, A is the leader and B is the follower.

Algorithm 1 For finding a schedule with $C_{max} \leq K$ if it exists.

Initialize $F_A(0) = F_B(0) = 0$;
for $u = 1, \ldots, m$ **do**
  for $v = 1, \ldots, m$ **do**
    Compute $L_A(u,v)$, $L_B(u,v)$, $S_A(u,v)$ and $S_B(u,v)$ via (1), (2), (3) and (4) respectively;
  end for
end for
for $v = 1, \ldots, m$ **do**
  Compute $F_A(v)$ and $F_B(v)$ via (5) and (6);
end for
if $F_A(m) < +\infty$ or $F_B(m) < +\infty$ **then**
  $C_{\max} \leq K$;
else
  $C_{\max} > K$.
end if

Given any feasible schedule, we can associate with it a decomposition of the m machines into *blocks*, each consisting of a maximal set of consecutive machines in which the two jobs are scheduled in the same order. We denote the block consisting of machines $M_u, M_{u+1}, \ldots, M_v$ as $<M_u, M_v>$ (see Figure 7). In a block, due to the s-precedence constraints, all the tasks of the leader job start at the same time. Given a block $<M_u, M_v>$, we can compute a number of quantities. (Assume for the moment that $v < m$.) If, in $<M_u, M_v>$, A is the leader, then we call *leader span* the length of the longest A-task in the block, and denote it with $L_A(u,v)$:

$$L_A(u,v) = \max_{u \leq j \leq v} \{p_j^A\}, \tag{1}$$

and similarly, if B is the leader, the leader span is given by:

$$L_B(u,v) = \max_{u \leq j \leq v} \{p_j^B\}. \tag{2}$$

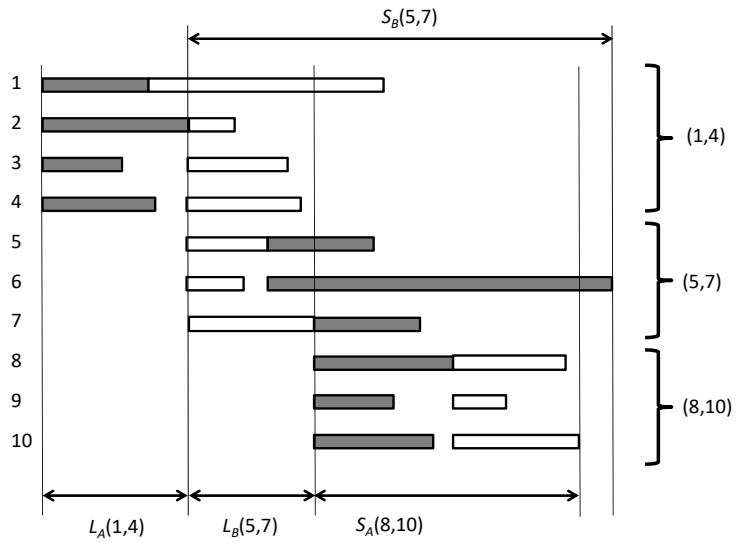

Figure 7. A sample schedule for $Fm|n = 2, s - prec|C_{\max}$. The tasks of job A are in grey.

Notice that, due to the definition of block, in the block that follows $<M_u, M_v>$, the roles of leader and follower are exchanged. Hence, the time at which the leader completes its longest task in $<M_u, M_v>$ is also the start time of the other job's tasks in the next block.

Given a block $<M_u, M_v>$, suppose again that A is the leader. We let $S_A(u, v)$ indicate the *span* of block $<M_u, M_v>$; i.e., the difference between the maximum completion time of a B-task and the start time of all A-tasks in $<M_u, M_v>$. This is given by:

$$S_A(u, v) = \max_{u \le j \le v} \{ \max_{u \le h \le j} \{p_h^A\} + p_j^B\}, \tag{3}$$

and exchanging the roles of leader and follower in $<M_u, M_v>$, we get

$$S_B(u, v) = \max_{u \le j \le v} \{ \max_{u \le h \le j} \{p_h^B\} + p_j^A\}. \tag{4}$$

Notice that trivial lower and upper bounds for the minimum makespan are given by

$$LB = \max\{ \max_{1 \le j \le m} \{p_j^A\}, \max_{1 \le j \le m} \{p_j^B\}\}$$

and

$$UB = \max_{1 \le j \le m} \{p_j^A\} + \max_{1 \le j \le m} \{p_j^B\}$$

respectively. In what follows we address the problem of determining a schedule having a makespan not larger than K, or prove that it does not exist. Assuming that all processing time values are integers, a binary search over the interval $[LB, UB]$ allows one to establish the value of the minimum makespan.

As we already observed, a relevant difference between $Fm||C_{max}$ and $Fm|s-prec|C_{max}$ is that, in a feasible schedule for $Fm|s-prec|C_{max}$, the value of C_{max} may not be attained on the last machine, but rather on any machine. This fact requires carefully handling by the algorithm.

Let $F_A(v)$ be the minimum sum of leader spans of all blocks from M_1 to M_v, when A is the leader of the last block (i.e., the block including M_v). Similarly, $F_B(v)$ is the same when B is the leader of the last block. In order to write a functional equation for $F_A(v)$ and $F_B(v)$, we introduce the notation $\delta(x) = 0$ if $x \le 0$ and $\delta(x) = +\infty$ if $x > 0$.

Hence, we write

$$F_A(v) = \min_{0 \le u \le v} \{F_B(u) + L_A(u + 1, v) + \delta(F_B(u) + S_A(u + 1, v) - K)\}. \tag{5}$$

The first terms accounts for the fact that in the previous block the leader is B, while the rightmost term ($\delta(\cdot)$) rules out solutions in which the sum of the start time of the last block and the span of the block itself exceeds K. Symmetrically, one has:

$$F_B(v) = \min_{0 \le u \le v} \{F_A(u) + L_B(u + 1, v) + \delta(F_A(u) + S_B(u + 1, v) - K)\}. \tag{6}$$

Expressions (5) and (6) are computed for $v = 1, \ldots, m$. If at least one of the values $F_A(m)$ and $F_B(m)$ has a finite value, a schedule of makespan not exceeding K exists. The values of machine index for which each minimum in (5) and (6) is attained define the blocks of the schedule, which can, therefore, be easily backtracked.

Equations (5) and (6) must be initialized, simply letting $F_A(0) = F_B(0) = 0$.

Notice that in general one cannot infer the value of the minimum makespan schedule directly from this procedure. If the minimum in the computation of $F_A(m)$ has been attained for, say, machine M_u, it does not imply that $F_B(u) + S_A(u + 1, m)$ is indeed the minimum makespan. This is because the overall

makespan may be due to a previous machine, and the algorithm has no control on this aspect. For instance, in the sample schedule of Figure 7 the makespan is attained on machine M_6. However, its actual value has no relevance, so long as it does not exceed K, since it does not affect the values $F_A(v)$ and $F_B(v)$ subsequently computed.

Concerning complexity, each computation of (5) and (6) requires $O(m)$ comparisons. Since the whole procedure is repeated at each step of a binary search over $[LB, UB]$, the following result holds.

Theorem 4. *Problem* $Fm|n = 2, s - prec|C_{max}$ *can be solved in* $O(m^2 log(UB - LB))$.

4. Further Research

In this paper we established some preliminary complexity results for perhaps the most basic cases of shop problems with s-precedence constraints. Here, we briefly elaborate on possible research directions.

- *Job shop problem with three jobs*. The job shop problem with more than two jobs is NP-hard. This is a direct consequence of the fact that $J|s-prec|C_{max}$ can be viewed as a generalization of $J||C_{max}$, which is NP-hard with three jobs [11].

Theorem 5. $J|n = 3, s - prec|C_{max}$ *is NP-hard.*

Proof. Consider an instance I of $J|n = 3|C_{max}$, in which O_i^k denotes the i-th task of job J^k in I, having processing time p_i^k on machine $M(O_i^k)$.

We can define an instance I' of $J|n = 3, s - prec|C_{max}$ with the same number of machines. The three jobs of I' are obtained replacing each task O_i^k of I with a sequence of two tasks $O_{i'}^k$ and $O_{i''}^k$, in which $O_{i'}^k$ precedes $O_{i''}^k$, $O_{i'}^k$ has length p_i^k and requires machine $M(O_i^k)$, while $O_{i''}^k$ has sufficiently small length $\epsilon > 0$ and also requires machine $M(O_i^k)$. As a consequence, in $J|s-prec|C_{max}$, the task $O_{i+1'}^k$ cannot start before $O_{i'}^k$ is started, but since $M(O_{i'}^k) = M(O_{i''}^k) = M(O_i^k)$, this can only occur after $O_{i'}^k$ is finished. So, for sufficiently small ϵ, a feasible schedule for I' having makespan $\leq K + m\epsilon$ exists if and only if a feasible schedule for I exists having makespan $\leq K$. □

Notice that the above reduction cannot be applied to $F|n = 3, s - prec|C_{\max}$, since in the flow shop each job visits all machines exactly once. In fact, the complexity of $Fm|s-prec|C_{\max}$ is open, even for fixed $m \geq 3$ or fixed $n \geq 3$.

- *Open problems with two jobs*. The approach in Section 2 for $J2|n = 2, s - prec|C_{\max}$ cannot be trivially extended to more than two machines. The complexity of this case is open. Additionally, an open issue is whether a more efficient algorithm can be devised for $J2|n = 2, s - prec|C_{\max}$, and a strongly polynomial algorithm for $Fm|n = 2, s - prec|C_{\max}$.

Author Contributions: Conceptualization, A.A. F.R. and S.S.; methodology, A.A. F.R. and S.S.; validation, A.A. F.R. and S.S.; formal analysis, A.A. F.R. and S.S.; investigation, A.A. F.R. and S.S.; resources, A.A. F.R. and S.S.; writing–original draft preparation, A.A. F.R. and S.S.; writing–review and editing, A.A. F.R. and S.S.; visualization, A.A. F.R. and S.S.; supervision, A.A. F.R. and S.S.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kim, E.-S.; Posner, M.E. Parallel machine scheduling with s-precedence constraints. *IIE Trans.* **2010**, *42*, 525–537. [CrossRef]
2. Kim, E.-S.; Sung, C.-S.; Lee, I.-S. Scheduling of parallel machines to minimize total completion time subject to s-precedence constraints. *Comput. Oper. Res.* **2009**, *36*, 698–710. [CrossRef]

3. Kim, E.-S. Scheduling of uniform parallel machines with s-precedence constraints. *Math. Comput. Model.* **2011**, *54*, 576–583. [CrossRef]
4. Tamir, T. Scheduling with Bully Selfish Jobs. *Theory Comput. Syst.* **2012**, *50*, 124–146. [CrossRef]
5. Demeulemeester, E.L.; Herroelen, W.S. *Project Scheduling, a Research Handbook*; Springer Science & Business Media: Berlin, Germany, 2006.
6. Agnetis, A.; Lucertini, M.; Nicolò, F. Flow Management in Flexible Manufacturing Cells with Pipeline Operations. *Manag. Sci.* **1993**, *39*, 294–306. [CrossRef]
7. Avella, P.; Boccia, M.; Mannino, C.; Vasilev, I. Time-indexed formulations for the runway scheduling problem. *Transp. Sci.* **2017**, *51*, 1031–1386. [CrossRef]
8. Briskorn, D.; Angeloudis, P. Scheduling co-operating stacking cranes with predetermined container sequences. *Discret. Appl. Math.* **2016**, *201*, 70–85. [CrossRef]
9. Bertel, S.; Billaut, J.-C. A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation, *Eur. J. Oper. Res.* **2004**, *159*, 651–662. [CrossRef]
10. Pinedo, M. *Scheduling, Theory, Algorithms and Systems*; Springer: Cham, Switzerland, 2018.
11. Brucker, P.; Sotskov, Y.N.; Werner, F. Complexity of shop scheduling problems with fixed number of jobs: A survey. *Math. Methods Oper. Res.* **2007**, *65*, 461–481. [CrossRef]