algorithms

MDPI

Article

Scheduling a Single Machine with Primary and Secondary Objectives

Nodari Vakhania

Centro de Investigación en Ciencias, Universidad Autonoma del Estado de Morelos, 62210 Cuernavaca, Mexico; nodari@uaem.mx

Received: 27 February 2018; Accepted: 31 May 2018; Published: 5 June 2018

Abstract: We study a scheduling problem in which jobs with release times and due dates are to be processed on a single machine. With the primary objective to minimize the maximum job lateness, the problem is strongly NP-hard. We describe a general algorithmic scheme to minimize the maximum job lateness, with the secondary objective to minimize the maximum job completion time. The problem of finding the Pareto-optimal set of feasible solutions with these two objective criteria is strongly NP-hard. We give the dominance properties and conditions when the Pareto-optimal set can be formed in polynomial time. These properties, together with our general framework, provide the theoretical background, so that the basic framework can be expanded to (exponential-time) implicit enumeration algorithms and polynomial-time approximation algorithms (generating the Pareto sub-optimal frontier with a fair balance between the two objectives). Some available in the literature experimental results confirm the practical efficiency of the proposed framework.

Keywords: scheduling single machine; release time; lateness; makespan; bi-criteria scheduling; Pareto-optimal solution

1. Introduction

We study a scheduling problem in which jobs with release times and due dates are to be processed by a single machine. The problem can be described as follows. There are n jobs j ($j = 1, ..., n$) and a single machine available from Time 0. Every job j becomes available at its release time r_j, needs continuous processing time p_j on the machine and is also characterized by the due date d_j, which is the desirable time moment for the completion of that job. These are are non-negative integral numbers. A feasible schedule S assigns to each job j the starting time $t_j(S)$, such that $t_j(S) \geq r_j$ and $t_j(S) \geq t_k(S) + p_k$, for any job k included earlier in S; the first inequality says that a job cannot be started before its release time, and the second one reflects the restriction that the machine can handle at most one job at a time. If job j completes behind its due date, i.e., $c_j = t_j(S) + p_j > d_j$, then it has a positive lateness $L_j = c_j - d_j$, otherwise its lateness is non-positive. Our primary objective is to find a feasible schedule in which the maximum job lateness is the minimal possible among all feasible schedules. However, we also consider a secondary objective to minimize the maximum job completion time or the makespan.

The problems involving both above objective criteria are strongly NP-hard, and the two objectives are, in general, contradictory; i.e., both of them cannot simultaneously be reached. Then, one may look for a subset of feasible solutions that are somehow acceptable with respect to both objective criteria. A Pareto-optimal frontier is constituted by a subset of all feasible solutions that are not dominated by some other feasible solution, in the sense that no other solution can surpass the former one with respect to both objective values. Finding Pareto-optimal frontier often remains NP-hard, as it is in our case. In particular, for the problems of finding among all feasible schedules with a given maximum job

lateness one with the minimum makespan, and vice versa, among all feasible schedules with a given makespan, finding one with the minimum maximum job lateness is strongly NP-hard (see Section 2).

According to the conventional three-field notation introduced by Graham et al., our primary problem with the objective to minimize maximum job lateness (the secondary one to minimize the makespan, respectively) is abbreviated as $1|r_j|L_{\max}$ ($1|r_j|C_{\max}$, respectively); here in the first field, the single-machine environment is indicated; the second field specifies the job parameters; and in the third field, the objective criterion is given. The problem $1|r_j|L_{\max}$ is known to be strongly NP-hard (Garey and Johnson [1]), whereas $1|r_j|C_{\max}$ is easily solvable by a greedy algorithm that iteratively includes any earliest released job at its release time or the completion time of the latest so far assigned job, whichever magnitude is larger. Although being strongly NP-hard, the former problem can be efficiently approximated. A venerable $O(n \log n)$ two-approximation algorithm that is commonly used for problem $1|r_j|L_{\max}$ was originally proposed by Jackson [2] for the version of the problem without release times and then was extended by Schrage [3] to take into account job release times. This heuristics, that is also referred to as the ED (Earliest Due date) heuristics, iteratively, at each scheduling time t (given by job release or completion time), among the jobs released by time t, schedules the one with the smallest due date. Let us note here that, in terms of the minimization of the makespan, the ED heuristics has an important advantage that it creates no machine-idle time that can be evaded. Potts [4] showed that by a repeated application of the heuristics $O(n)$ times, the performance ratio can be improved to $3/2$ resulting in an $O(n^2 \log n)$ time performance. Later, Hall and Shmoys [5] illustrated that the application of the ED heuristics to the original and a specially-defined reversed problem may lead to a further improved approximation of $4/3$. Nowicki and Smutnicki [6] have shown that the approximation ratio $3/2$ can also be achieved in time $O(n \log n)$. In practice, the ED heuristics turned out to be the most efficient fast solution method for problem $1|r_j|L_{\max}$, far better than one would suggest based on the above theoretical worst-case performance ratio (see Larson et al. [7], Kise et al. [8] and Vakhania et al. [9]). We shall reveal the benefit of such performance for our bi-criteria scheduling problem later.

As for the exact solution methods for problem $1|r_j|L_{\max}$, the branch-and-bound algorithm with good practical behavior was proposed in McMahon and Florian [10] and Carlier [11] (the practical behavior of these two algorithms was compared also more recently by Sadykov and Lazarev [12]). Two more exact implicit enumerative algorithms were proposed, in Grabowski et al. [13] and Larson et al. [14]. More recently, Pan and Shi [15] and Liu [16] presented other branch-and-bound algorithms (in the latter reference, the version with precedence constraints was studied). The preemptive setting $1|r_j, pmtn|L_{\max}$ is easily solved by the preemptive version of Jackson's extended heuristics, as it always interrupts non-urgent jobs in favor of a newly-released more urgent one. In fact, the preemptive ED heuristics is also useful for the solution of the non-preemptive version, as it gives a strong lower bound for the latter problem (see, for example, Gharbi and Labidi [17] for a recent study on this subject).

The version in which a feasible schedule may not include any idle time interval, abbreviated $1|r_j, nmit|L_{\max}$, is strongly NP-hard according to Lenstra et al. [18] (minimization of the makespan and that of the length of idle time intervals are closely related objectives). The problem admits the same approximation as the unrestricted version; as Kacem and Kellerer [19] have observed, job release times can be adopted so that the ED heuristics and Potts's above-mentioned extension maintain the same approximation ratio for problem $1|r_j, nmit|L_{\max}$ as for $1|r_j|L_{\max}$. Hoogeveen has considered the no machine idle time version in the bi-criteria setting. Instead of minimizing the lateness, he has introduced the so-called target start time s_j of job j; s_j is the desirable starting time for job j, similarly to the due date d_j being the desirable completion time for job j. Then, besides the minimization of the maximum job lateness, the maximum job promptness (the difference between the target and real start times of that job) can also be minimized. The above paper considers the corresponding bi-criteria scheduling problem and finds the Pareto-optimal set of feasible solutions. Lazarev [20] and Lazarev et al. [21] have proposed a polynomial time solution finding the Pareto-optimal set for

two special cases of our bi-criteria problem with specially-ordered job parameters and equal-length jobs, respectively. An exact enumerative algorithm for the no idle time problem with the objective to minimize maximum job lateness was proposed by Carlier et al. [22], whereas the practical importance of the solutions with no idle time intervals was emphasized by Chrétienne [23].

In general, in an optimal solution to problem $1|r_j|L_{\max}$, some idle time intervals might be unavoidable (a non-urgent job has to wait until a more urgent one gets released; hence, the corresponding idle time interval arises): a problem instance might easily be constructed possessing an optimal solution to $1|r_j|L_{\max}$, but with no feasible solution for $1|r_j, nmit|L_{\max}$ (see Section 2).

In this paper, based on the study of the properties of the schedules generated by the ED heuristics and the ties of the scheduling problem $1|r_j|L_{\max}$ with a version of the bin packing problem, we propose a general solution method that is oriented toward both objective criteria, the maximum job lateness and the maximum job completion time (the makespan). Considering a single criterion problem, we first observe that different feasible solutions with different makespans may posses the same maximum job lateness. Though, as we will see, finding one with the minimum makespan is NP-hard. Viewing the problem in light of the bin packing problem with different bin capacities, we show how the known heuristic algorithms with reasonably good approximations can be adopted for the solution of the bi-criteria problem.

We establish the conditions when the Pareto-optimal frontier with the two objective functions (the maximum lateness and the makespan) can be formed in polynomial time. This, together with a general algorithmic framework that we present, provides a method for the construction of (exponential-time) implicit enumeration algorithms and polynomial-time approximation algorithms (the latter approximation algorithms generate a Pareto-sub-optimal frontier with a "fair balance" between the two objectives).

Our framework applies the partition of the scheduling horizon into two types of segments containing urgent and non-urgent jobs, respectively. Intuitively, segments with urgent jobs are to occupy quite restricted time intervals, whereas the second type of segments can be filled out, in some optimal fashion, with non-urgent jobs. The urgency of jobs is determined based on the first objective criterion, i.e., such jobs may potentially realize the maximum lateness in the optimal schedule S^{opt} for problem $1|r_j|L_{\max}$. We determine the time intervals within which a sequence of urgent jobs is to be scheduled. We refer to such sequences as kernels and to the intervals in between them as bins. Our scheme, iteratively, fills in the bin intervals by the non-urgent jobs; it determines the corresponding kernel segments before that. The less gaps are left within the bin intervals, the less is the resultant maximum job completion time, whereas the way the bin intervals are filled out is reflected also by the maximum job lateness.

We exploit the observation that the total idle time interval in a bin of a given feasible schedule can be reduced without increasing the maximum job lateness of that schedule. This kind of "separate" scheduling turns out to be helpful in the search for a compromise between the two objectives. In practice, analyzing the computational experiments for the problem $1|r_j|L_{\max}$ reported in [9], we may assert that, by incorporating the ED heuristics into our scheme, a balanced distribution of the jobs within the bins imposing almost neglectable delays for the kernel jobs can be obtained. For the vast majority of the tested instances in [9], the maximum job lateness is very close to the optimum. At the same time, since the ED heuristics creates no avoidable machine idle time, the minimum makespan is achieved. We discuss this issue in more detail in Section 5.

In Section 2, we study our two objective functions and establish the time complexity of related Pareto-optimal problems. In Section 3, we give necessary preliminaries to our algorithmic scheme. In Section 4, we describe our method for partitioning the whole scheduling horizon into urgent and non-urgent zones. In Section 5, we introduce the binary search procedure that is used to verify the existence of a feasible solution with some threshold lateness and describe the rest of the proposed framework. In Section 6, we give final remarks.

2. Complexity of the Bi-Objective Problem

In this section, we will see why finding the Pareto-optimal frontier of feasible solutions for our bi-criteria scheduling problem is NP-hard. In the Introduction, we gave an intuitive description of the Pareto-optimality concept (for a detailed guideline on multi-objective optimization, the reader may have a look at, for example, Ehrgott [24]). More formally, in a bi-criteria optimization problem, two objective functions f_1 and f_2 over the set $\mathcal{F}$ of feasible solutions is given. These functions might be mutually contradictory; hence, there may exist no feasible solution minimizing both objective functions simultaneously (considering the minimization version). More precisely, if F_i^* is the optimal value of a single-criterion problem with the objective to minimize function f_i, then there may exist no feasible solution to the problem that attains simultaneously both optimal values F_1^* and F_2^*. In this situation, it is reasonable to look for feasible solutions that are somehow acceptable for both the objective functions. A commonly-used dominance relation is defined as follows.

Solution $\sigma_1 \in \mathcal{F}$ dominates solution $\sigma_2 \in \mathcal{F}$ if $f_i\sigma_1 < f_i\sigma_2$, for $i = 1, 2$; in fact, we allow $\leq$, instead of $<$ for either $i = 1$ or $i = 2$, and require having at least one strict inequality.

Now, we refer to $\sigma \in \mathcal{F}$ as a Pareto-optimal solution if no other solution from set $\mathcal{F}$ dominates it, and we call the set of all such solutions the Pareto-optimal frontier of feasible solutions. Forming a Pareto-optimal frontier of feasible solutions may not be easy; for instance, the corresponding single-objective problems may already be intractable (clearly, the solution of a bi-objective setting requires the solution of the corresponding single-objective settings).

Consider now the two basic objective functions dealt with in this paper, the maximum job lateness and the makespan. The two corresponding single-objective problems are $1|r_j|L_{\max}$ and $1|r_j|C_{\max}$.

The two objective criteria are contradictory, i.e., minimizing the maximum job lateness $L_{\max}$ does not necessarily minimize the makespan $C_{\max}$, and vice versa: it can be easily seen that an optimal schedule minimizing $L_{\max}$ may contain avoidable gaps and hence may not minimize $C_{\max}$. For example, consider an instance of $1|r_j|L_{\max}$ with two jobs with $r_1 = 0, r_2 = 3, p_1 = p_2 = 5, d_1 = 20, d_2 = 9$. In the schedule minimizing $L_{\max}$, there is a gap $[0, 3)$; Job 2 is included at its release time and is followed by Job 1, whereas in the schedule minimizing $C_{\max}$, Job 1 starts at its release time and is followed by Job 2. The maximum job lateness and the makespan in the first and the second schedules are -1 and 13 (respectively) and 1 and 10 (respectively). Below, we state related Pareto-optimality problems.

Problem 1. *Among all feasible schedules with a given makespan, find one with the minimum maximum job lateness.*

Problem 2. *Among all feasible schedules with a given maximum job lateness, find one with the minimum makespan.*

Clearly, Problem 1 is strongly NP-hard since, as earlier mentioned, problem $1|r_j|L_{\max}$ is strongly NP-hard [1]. On the other hand, problem $1|r_j|C_{\max}$ can easily be solved by the following linear-time list scheduling algorithm, which leaves no idle time interval that can be avoided: at every scheduling time, stating from the minimum job release time, include any released job, update the current scheduling time as the maximum between the completion time of that job and the next minimum job release time of an unscheduled job, and repeat the procedure until all jobs are scheduled (a modification of this greedy algorithm that at each scheduling time, among the released jobs, includes one with the minimum due date is the ED heuristics mentioned in the Introduction). This greedy solution also provides a polynomial-time solution for the decision version of problem $1|r_j|C_{\max}$ in which one wishes to know if there exists a feasible schedule with some fixed makespan M. Indeed, if M is less than the optimum, then clearly, the answer is "no". If M is greater than or equal to the minimum makespan M^*, then the answer is "yes": we can introduce in a schedule with the makespan M^* gaps with a total length $M - M^*$, appropriately, so that the resultant makespan will be exactly M.

To show that Problem 2 is strongly NP-hard, we prove that its decision version is strongly NP-complete. Let M be some threshold value for the makespan. Without loss of generality, we assume that this threshold is attainable, i.e., there is a feasible schedule with the objective value M. Then, the decision version of Problem 2 can be formulated as follows:
Problem 2-D. For an instance of problem $1|r_j|L_{\max}$ (the input), among all feasible schedules with a given maximum job lateness, does there exist one with the makespan not exceeding magnitude M (here, the output is a "yes" or a "no" answer)?

Theorem 1. *Problem 2-D is strongly NP-complete.*

Proof. To begin with, it is easy to see that the problem is in class NP. First, we observe that the size of an instance of problem $1|r_j|L_{\max}$ with n jobs is $O(n)$. Indeed, every job j has three parameters, r_j, p_j and d_j; let M be the maximum such magnitude, i.e., the maximum among all job parameters. Then, the number of bits to represent our problem instance is bounded above by $3n \log M$. Since constant M fits within a computer word, the size of the problem is $O(n)$. Now, given a feasible schedule with some maximum job lateness, we can clearly verify its makespan in no more than n steps, which is a polynomial in the size of the input. Hence, Problem 2-D belongs to class NP.

Next, we use the reduction from a strongly NP-complete three-PARTITION problem to show the NP-hardness of our decision problem. In three-PARTITION, we are given a set A of $3m$ elements, their sizes $C = \{c_1, c_2, \ldots, c_{3m}\}$ and an integer number B with $\sum_{i=1}^{3m} c_i = mB$, whereas the size of every element from set A is between $B/4$ and $B/2$. We are asked if there exists the partition of set A into m (disjoint) sets $A_1, \ldots, A_m$ such that the size of the elements in every subset sums up to B.

Given an arbitrary instance of three-PARTITION, we construct our scheduling instance with $3m + m$ jobs with the total length of $\sum_{\iota=1}^{n} c_\iota + m$ as follows. We have $3m$ partition jobs $1, \ldots, 3m$ with $p_i = c_i$, $r_i = 0$ and $d_i = 2mB + m$, for $i = 1, \ldots, 3m$. Besides, we have m separator jobs $j_1, \ldots, j_m$ with $p_{j_\iota} = 1$, $r_{j_\iota} = B\iota + \iota - 1$, $d_{j_\iota} = B\iota + \iota$. Note that this transformation creating a scheduling instance is polynomial as the number of jobs is bounded by the polynomial in m, and all magnitudes can be represented in binary encoding in $O(m)$ bits.

First, we easily observe that there exist feasible schedules in which the maximum job lateness is zero: we can just schedule the separator jobs at their release times and include the partition jobs arbitrarily without pushing any separator job. Now, we wish to know whether among the feasible schedules with the maximum job lateness of zero there is one with makespan $mB + m$. We claim that this decision problem has a "yes" answer iff there exists a solution to three-PARTITION.

In one direction, suppose three-PARTITION has a solution. Then, we schedule the partition jobs corresponding to the elements from set A_1 within the interval $[0, B]$ and partition jobs from set A_i within the interval $[(i-1)B + i - 1, iB + i - 1]$, for $i = 2, \ldots, m$ (we schedule the jobs from each group in a non-delay fashion, using, for instance, the ED heuristics). We schedule the separator job j_i within the interval $[iB + i - 1, iB + i]$, for $i = 1, \ldots, m$. Therefore, the latest separator job will be completed at time $mB + m$, and the makespan in the resultant schedule is $mB + m$.

In the other direction, suppose there exists a feasible schedule S with makespan $mB + m$. Since the total sum of job processing times is $mB + m$, there may exist no gap in that schedule. However, then, the time interval of length B before every separator job must be filled in completely with the partition jobs, which obviously gives a solution to three-PARTITION. □

3. Basic Definitions and Properties

This section contains necessary preliminaries to the proposed framework including some useful properties of the feasible schedules created by the ED heuristics (we call them ED-schedules). Our approach is based on the analysis of the properties of ED-schedules. This analysis is initially carried out for the decision version of single-criterion problem $1|r_j|L_{\max}$: we shall particularly be

interested in ED-schedules with the maximum job lateness not exceeding a given threshold (so our analysis is carried out in terms of the minimization of the maximum job lateness). First, we give a detailed description of the ED heuristics.

The heuristics distinguishes n scheduling times, the time moments at which a job is assigned to the machine. Initially, the earliest scheduling time is set to the minimum job release time. Among all jobs released by that time, a job with the minimum due date is assigned to the machine (ties being broken by selecting the longest job). Iteratively, the next scheduling time is either the completion time of the latest job assigned so far to the machine or the minimum release time of an unassigned job, whichever is more (no job can be started before the machine becomes idle, nor can it be started before it gets released). Among all jobs released by every newly- determined (as just specified) scheduling time, a job with the minimum due date is assigned to the machine. Note that since there are n scheduling times and at each scheduling time, a minimal element in an ordered list is searched for, the time complexity of the heuristics is $O(n \log n)$.

A gap in an ED-schedule is its maximal consecutive time interval in which the machine is idle (with our convention, there occurs a zero-length gap (c_j, t_i) whenever job i starts at its release time immediately after the completion of job j).

A block in an ED-schedule is its consecutive part consisting of the successively scheduled jobs without any gap in between, which is preceded and succeeded by a (possibly a zero-length) gap.

Suppose job i preceding job j in ED-schedule S is said to push job j in S if j will be rescheduled earlier whenever i is forced to be scheduled behind j (it follows that jobs i and j belong to the same block).

Example 1. *We have seven jobs in our first problem instance. The processing times, the release times and the due dates of these jobs are defined as follows:*
$p_1 = p_3 = 5, p_2 = 30, p_4 = p_5 = p_6 = p_7 = 10.$
$r_3 = 11, r_5 = r_6 = r_7 = 42$, *whereas the rest of the jobs are released at Time 0, except Job 4, released at Time 36.*
$d_1 = 75, d_2 = 80, d_3 = 11, d_4 = 53, d_5 = d_6 = d_7 = 52.$

It can be readily verified that the ED heuristics will assign the jobs in increasing order of their indexes creating an ED-schedule $S = (1,0)(2,5)(3,35)(4,40)(5,50)(6,60)(7,70)$, *as depicted in Figure 1 (in every pair in brackets, the first number is the job index, and the second number is its starting time). In schedule S, consisting of a single block (there is no gap in it), Job 2 pushes Job 3 and the succeeding jobs.*

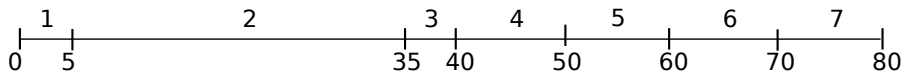

Figure 1. The initial Earliest Due date (ED)-schedule S for the problem instance of Example 1.

Given an ED-schedule S, let i be a job that realizes the maximum job lateness in that schedule, i.e., $L_i(S) = \max_j\{L_j(S)\}$, and let B be the block in S containing job i. Among all jobs $i \in B$ with $L_i(S) = \max_j\{L_j(S)\}$, the latest scheduled one is said to be an overflow job in schedule S. Clearly, every schedule contains at least one overflow job, whereas every block in schedule S may contain one or more overflow jobs (two or more overflow jobs from the same block will then be "separated" by non-kernel jobs).

A kernel in schedule S is the longest sequence of jobs ending with an overflow job o, such that no job from this sequence has a due date more than d_o.

We shall normally use $K(S)$ for the earliest kernel in schedule S, and abusing the terminology, we shall also use $K(S)$ for the set of jobs in the sequence. For kernel K, let $r(K) = \min_{i \in K}\{r_i\}$.

Note that every kernel is contained within some block in schedule S, and hence, it may contain no gap. The following observation states a sufficient optimality condition for the single-criterion problem $1|r_j|L_{\max}$ (the reader may have a look at [25] for a proof):

Observation 1. *The maximum job lateness in a kernel K cannot be reduced if the earliest scheduled job in K starts at time $r(K)$. Hence, if an ED-schedule S contains a kernel with this property, it is optimal for problem $1|r_j|L_{\max}$.*

Thus, we may ensure that there exists no feasible schedule with the value of our objective function less than that in solution S whenever the condition in Observation 1 is met. Then, we immediately proceed to the reconstruction of solution S aiming at the minimization of our second objective function, as we describe in the next section.

Suppose the condition in Observation 1 is not satisfied. Then, there must exist job e with $d_e > d_o$ (i.e., less urgent than the overflow job o), scheduled before all jobs of kernel K imposing a forced delay for the jobs of that kernel. We call job e an emerging job for kernel K. Note that an emerging job e and kernel K belong to the same block.

If the earliest scheduled job of kernel K does not start at its release time, it is immediately preceded and pushed by an emerging job, which we call the delaying (emerging) job for kernel K.

We can easily verify that in ED-schedule S of Example 1 (see Figure 1), there is a single kernel $K_1 = K(S)$ consisting of a single (overflow) Job 3, with $L_{\max}(S) = L_{3,S} = 40 - 11 = 29$ (note that $L_{7,S} = 80 - 52 = 28$). There are two emerging Jobs 1 and 2 for kernel K, and Job 2 is the delaying emerging job for that kernel.

Clearly, the maximum job lateness in schedule S may only be decreased by restarting the jobs of kernel $K = K(S)$ earlier. Note that if we reschedule an emerging job e behind the jobs of kernel K, we may restart these jobs earlier. As a result, the maximum job lateness in kernel K may be reduced. We shall refer to the operation of the rescheduling of emerging job e after kernel K as the activation of that emerging job for kernel K. To provide the early restarting for jobs of kernel K, we maintain any job scheduled behind kernel K in S scheduled behind kernel K (all these jobs are said to be in the state of activation for that kernel). We call the resultant ED-schedule a complementary to the S schedule and denote it by S_e. In general, a complementary schedule S_E is defined for a set of emerging jobs E in schedule S. In schedule S_E, all jobs from set E and all the jobs scheduled behind kernel K in S are in the state of activation for kernel K. It is easy to see that by activating a single emerging job e, kernel K will already be restarted earlier; hence, the maximum job lateness realized by a job in kernel K will be reduced.

Observation 2. *Suppose the release time of all the emerging jobs in a set E is increased to the magnitude $r(K)$, and the ED heuristics is newly applied to the modified instance. Then, if some job j scheduled behind kernel K in schedule S gets rescheduled before kernel K, $d_j > d_o$, where o is the corresponding overflow job in kernel K.*

Proof. By the ED heuristics, any job i released before the jobs of kernel K with $d_i \leq d_o$ would have been included within kernel K in schedule S. Hence, if a job scheduled behind kernel K in schedule S becomes included before kernel K, then $d_j > d_o$ (i.e., it is less urgent than all kernel jobs). □

By the above observation, if we merely increase the release time of any above available job j with $d_j > d_o$ and that of any emerging job from set E to magnitude $r(K)$, then the ED heuristics will create complementary schedule S_E for the modified instance.

Observation 3. *Let l be the delaying emerging job in schedule S. Then, in complementary schedule S_l, the earliest scheduled job of kernel $K = K(S)$ starts at its release time $r(K)$ and is preceded by a gap.*

Proof. By the definition of complementary schedule S_l, no job j with $d_j > d_o$ may be included before kernel $K(S)$ in schedule S_l. As we have already observed, no available job i with $d_i \leq d_o$ may be included before kernel K in complementary schedule S_l. Then, clearly, the time interval before the earliest released job of kernel K will remain idle in complementary schedule S_l, and the observation follows. □

For the problem instance of Example 1, we form set E with a single emerging Job 1 and obtain a complementary schedule $S_1 = (2,0)(3,30)(1,35)(4,40)(5,50)(6,60)(7,70)$, as depicted in Figure 2. There is a single kernel $K_2 = K(S_1)$ consisting of Jobs 5, 6 and 7 with $L_{\max}(S_1) = L_{7,S_1} = 80 - 52 = 28$ in schedule S_1 (note that $L_{3,S_1} = 35 - 11 = 24$).

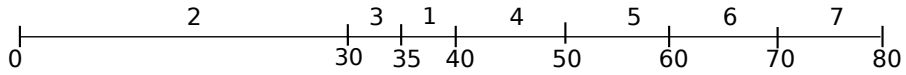

Figure 2. Complementary ED-schedule S_1 for Example 1.

The proof of our next observation can be seen in [26].

Property 1. *Suppose in schedule S_E there arises a (possibly zero-length) gap before kernel $K = K(S)$. Then, that schedule is optimal for problem $1|r_j|L_{\max}$ if $K(S_E) = K(S)$.*

Note that the above property yields another halting condition for problem $1|r_j|L_{\max}$. The following example illustrates that activation of an emerging job does not always yield an improvement in terms of the minimization of the maximum job lateness.

Example 2. *Now, we form an instance with six jobs as follows. Jobs 4 and 6 are kernel jobs, each defining a distinct kernel with the delaying emerging Jobs 3 and 5, respectively. Jobs 1 and 2 are other emerging jobs. All emerging jobs are released at Time 0, whereas kernel Jobs 4 and 6 are released at Times 20 and 50 respectively; $p_1 = p_2 = p_4 = p_6 = 5$, $p_3 = 25$, $p_5 = 20$. The due dates of emerging jobs are large enough numbers, Jobs 1 and 2 being slightly more urgent than the delaying emerging Jobs 3 and 5; $d_4 = 25$ and $d_6 = 55$. Then ED heuristics will deliver solution $S = (1,0)(2,5)(3,10)(4,35)(5,40)(6,60)$ with the delaying emerging Job 3. The lateness of Job 4 is 15, and that of Job 6 is 10; Job 4 forms the first kernel with the only overflow Job 4. In the complementary schedule $S_4 = (1,0)(2,5)(4,20)(3,25)(5,50)(6,70)$, there arises a gap $(10,20)$. In that schedule, a new kernel is formed by Job 6, which completes at Time 75 and has the lateness of 20. This example suggests that, instead of creating schedule S_1, it could have been better to consider schedule S_E, for a properly selected set E.*

4. Partitioning the Scheduling Horizon

In this section, we start the construction of a bridge between the two scheduling problems $1|r_j|L_{\max}$ and $1|r_j|C_{\max}$ establishing a helpful relationship between these problems and a version of a well-studied bin packing problem. To begin with, we introduce our bins. The notion of a bin is closely tied to that of a kernel. Given an ED-schedule with a set of kernels, we define the bins in that ED-schedule as the segments or the intervals left outside the kernel intervals. The bins will be packed with non-kernel jobs, and since, the interval within which every kernel can be scheduled has some degree of flexibility, the bins are defined with the same degree of flexibility. Abusing the terminology, we will use the term "bin" for both the time interval and the corresponding schedule portion, as well. Note that there is a bin between two adjacent kernel intervals and a bin before the first and after the last kernel interval; any kernel defines two neighboring bins adjacent to that kernel, one preceding and one succeeding that kernel.

We first describe how we obtain a partition of the scheduling horizon into the kernel and bin intervals. In the next section, we explain how this partition is used for the solution of our bi-criteria scheduling problem that is reduced to the distribution of non-kernel jobs within the bins in some (near) optimal fashion (that takes into account both objective criteria).

The initial partition is constructed based on the earliest kernel identified in the initial ED-schedule (one constructed for the originally given problem instance). This kernel already defines the corresponding two bins in that initial partition. For instance, in schedule S of Example 1 (Figure 1), kernel $K_1 = K(S)$ is detected, which already defines the two bins in that schedule. In the next generated

complementary schedule S_1 (Figure 2), there arises a new kernel within the second bin. Note that the execution interval of kernel K_1 in schedules S and S_1 is different (it is $[35, 40)$ in schedule S and $[30, 35)$ in schedule S_1). Correspondingly, the first bin is extended up to time moment 35 in schedule S, and it is extended up to Time 30 in schedule S_1. The newly arisen kernel $K_2 = K(S_1)$ defines two new bins in schedule S_1; hence, that schedule contains three bins, in total.

In general, the starting and completion times of every bin are not a priori fixed; they are defined in accordance with the allowable flexibility for the corresponding kernel intervals. Our scheme proceeds in a number of iterations. To every iteration, a complete schedule with a particular distribution of jobs into the bins corresponds, whereas the set of jobs scheduled in every kernel interval remains the same. In this way, two or more complete schedules for the same partition might be created (for instance, schedules S_1 and $(S_1)_1$ of Example 1; see Figures 2 and 3). At an iteration h, during the scheduling of a bin, a new kernel may arise, which is added to the current set of kernels $\mathcal{K}$ (kernel K_2 arises within the second bin in complementary schedule S_1 of Example 1): the updated set contains all the former kernels together with the newly arisen one (since schedule S of Figure 1 contains only one kernel, $\mathcal{K} = \{K_1\}$, there are only two bins in it; the next generated schedule S_1 of Figure 2 contains already two kernels K_1 and K_2, $\mathcal{K} = \{K_1, K_2\}$ and three bins). Note that the partition of the scheduling horizon is changed every time a new kernel arises.

Adjusting time intervals for kernel and bin segments: Although the time interval within which each kernel can be scheduled is restricted, it has some degree of flexibility, as we have illustrated in Example 1. In particular, the earliest scheduled job of a kernel K might be delayed by some amount without affecting the current maximum job lateness. Denote this magnitude by $\delta(K)$. Without loss of generality, the magnitude $\delta(K)$ can take the values from the interval $\delta(K) \in [0, p_l]$, where l is the delaying emerging job for kernel K in the initially constructed ED-schedule σ. Indeed, if $\delta(K) = 0$, the kernel will not be delayed, whereas $\delta(K) = p_l$ corresponds to the delay of the kernel in the initial ED-schedule S (which is 19 for kernel K_1 in schedule S of Figure 1). In particular, we let p_{max} be the maximum job processing time and shall assume, without loss of generality, that $\delta(K) \leq p_{max}$.

To define $\delta(K)$, let us consider a partial schedule constructed by the ED heuristics for only jobs of kernel $K \in \mathcal{K}$. The first job in that schedule starts at time $r(K)$ (we assume that there is no emerging job within that sequence, as otherwise we continue with our partitioning process). Note that the lateness of the overflow job of that partial schedule is a lower bound on the maximum job lateness; denote this magnitude by $L^*(K)$. Then, $L^* = \max_{K \in \mathcal{K}}\{L^*(K)\}$ is also a lower bound on the same objective value. Now, we let $\delta(K) = L^* - L^*(K)$, for every kernel $K \in \mathcal{K}$. The following observation is apparent:

Observation 4. *For $\delta(K) = 0$, the lateness of the latest scheduled job of kernel K is a lower bound on the optimal job lateness, and for $\delta(K) = p_l$, it is a valid upper bound on the same objective value.*

Observation 5. *In a schedule S_{opt} minimizing the maximum job lateness, every kernel K starts either no later than at time $r(K) + \delta(K)$ or it is delayed by some $\delta \geq 0$, where $\delta \in [0, p_{max}]$.*

Proof. First note that in any feasible schedule, every kernel K can be delayed by the amount $\delta(K)$ without increasing the maximum job lateness. Hence, kernel K does not need to be started before time $r(K) + \delta(K)$. Furthermore, in any created ED-schedule, the delay of any kernel cannot be more than p_{max}, as for any delaying emerging job l, $p_l \leq p_{max}$. Hence, $\delta \in [0, p_{max}]$. □

Thus for a given partition, the starting and completion time of every bin in a corresponding complete schedule may vary according to Observation 5. Our scheme incorporates a binary search procedure in which the trial values for δ are drown from interval $[0, p_{max}]$. To every iteration in the binary search procedure, some trial value of δ corresponds, which determines the maximum allowable job lateness, as we describe below.

We observe that not all kernels are tight in the sense that, for a given kernel $K \in \mathcal{K}$, $\delta(K)$ might be strictly greater than zero. By Observation 5, in any generated schedule with trial value δ, every kernel

K is to be started within the interval $[r(K)+\delta(K), r(K)+\delta(K)+\delta]$, the corresponding bin intervals being determined with respect to these starting times. Notice that if the latest job of kernel K completes at or before time moment $r(K)+P(K)+\delta$, then the lateness of no job of that kernel will surpass the magnitude $L(\delta) = r(K)+P(K)+\delta-d$, where d is the due date of the corresponding overflow job and $P(K)$ is the total processing time of the jobs in kernel K.

We shall refer to the magnitude $L(\delta) = r(K)+P(K)+\delta-d$ as the threshold on the maximum allowable job lateness corresponding to the current trial value δ. Ideally, not only jobs of kernel K, but no other job is to have the lateness more than $L(\delta)$. Our scheme verifies if there exists a feasible schedule with the maximum job lateness no more than $L(\delta)$. If it succeeds (fails, respectively) to create a feasible schedule with the maximal job lateness no more than threshold $L(\delta)$, the next smaller (larger, respectively) value of δ is taken in the binary partition mode. According to our partition, the lateness of no kernel job may surpass the threshold $L(\delta)$. Hence, it basically remains to schedule the bin intervals (see the next section). As already noted, if while scheduling some bin, the next incoming job results in a lateness greater than $L(\delta)$, then a new kernel K including this job is determined (see the next section for the details). Then, the current set of the kernels is updated as $\mathcal{K} := \mathcal{K} \cup \{K\}$, and the scheme proceeds with the updated partition.

A positive delay for a kernel might be unavoidable in an optimal job arrangement. For instance, the time interval of kernel K_1 in schedules S and S_1 (Figures 1 and 2) is $[35, 40)$ and $[30, 35)$, respectively. The time interval of kernel K_1 remains to be $[30, 35)$ in the optimal solution. Indeed, let us form our next set E again with the same emerging Job 1 constructing complementary schedule $(S_1)_1 = (2,0)(3,30)(4,36)(5,46)(6,56)(7,66)(1,76)$; see Figure 3 (this complementary schedule has a gap $[35, 36)$ marked as a dark region in the figure).

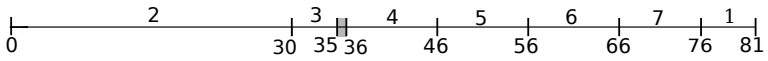

Figure 3. A complementary ED-schedule $(S_1)_1$.

It is easily verified that schedule $(S_1)_1$ with kernels K_1 and K_2 and the corresponding three arranged bins is optimal with $L_{\max}((S_1)_1) = L_{3,(S_1)_1} = 35-11 = 24 = L_{7,(S_1)_1} = 76-52 = 24$. In that schedule, we have achieved an optimal balance between the delays of the two kernels by redistributing the available jobs within the bins. As a result, the makespan is reduced without affecting the maximum job lateness. Note that the jobs of kernel K_1 and also those of kernel K_2 are delayed by an appropriate amount of time units without affecting the maximum job lateness. Our framework determines such an appropriate delay for every detected kernel, resulting in an optimal balance between kernel and bin intervals. Observe that, although the partition of the scheduling horizon in schedules S_1 and $(S_1)_1$ is the same, the bins are packed in different ways in these two complementary schedules.

Instead of forming the complementary schedules, we shall construct a direct algorithm for scheduling the bins in the following section.

5. Scheduling the Bins

At the iteration in the binary search procedure with the trial value δ and a given partition, we have reduced the scheduling problem to that of filling in the bin intervals by non-kernel jobs. We schedule the bins according to the order in which they appear in the current partition of bin and kernel segments. If all the bins from the current partition are scheduled without surpassing current threshold value $L(\delta)$ (see the previous section), these bins together with the corresponding kernels are merged respectively, forming the overall complete schedule (we have the successful outcome for trial δ). The created feasible solution respecting threshold $L(\delta)$ will have an optimal or sub-optimal makespan among all feasible solutions respecting $L(\delta)$ (see Theorem 2 and Lemma 3).

In this section, we describe how the bins can efficiently be filled in with non-kernel jobs and how during this process, new kernels and the corresponding bins may be declared, yielding an updated

partition of the scheduling horizon. While scheduling the bins, our (secondary) goal is to reduce the length of the total idle time interval, which yields a reduced makespan. Although most bin scheduling problems are NP-hard, there are heuristic algorithms with good approximation for these problems. Adopting such a heuristics for our scheduling model, we may pack our bins with close to the minimal total gap length (makespan), as we describe below. Whenever strict optimality is important, an implicit enumerative algorithm can be applied. First, we need a few additional notions.

Let B be the bin immediately preceding kernel K (our construction is similar for the last bin of the current partition). While scheduling bin B, we distinguish two types of the available jobs for that bin. We call jobs that may only be feasibly scheduled within that bin y-jobs and ones that may also be feasibly scheduled within a succeeding bin the x-jobs. We have two sub-types of the y-jobs for bin B: a Type (a) y-job may only be feasibly scheduled within that bin, unlike the Type (b) y-jobs, which could have also been included in some preceding bin (a Type (b) y-job is released before the interval of bin B and can potentially be scheduled within a preceding bin).

The procedure for scheduling bin B has two phases: Phase 1 consists of two passes. In the first pass, y-jobs are scheduled. In the second pass, x-jobs are included within the remaining available space of the bin by a specially-modified decreasing next fit heuristics, commonly used for bin packing problems. Note that all Type (a) y-jobs can be feasibly scheduled only within bin B. Besides Type (a) y-jobs, we are also forced to include all the Type (b) y-jobs in bin B, unless we carry out a global job rearrangement and reschedule a Type (b) y-job within some preceding bins. We consider such a possibility in Phase 2.

Phase 1, first pass (scheduling y-jobs): The first pass of Phase 1 consists of two steps. In Step 1, the y-jobs are scheduled in bin B by the ED heuristics resulting in a partial ED-schedule, which we denote by $S(B, y)$. It consists of all the y-jobs that the ED heuristics could include without surpassing the current threshold $L(\delta)$ (recall that $L(\delta)$ is the currently maximum allowable lateness).

If schedule $S(B, y)$ is y-complete, i.e., it contains all the y-jobs for bin B, then it is the output of the first pass. Otherwise, the first pass continues with Step 2. We need the following discussion before we describe that step.

If schedule $S(B, y)$ is not y-complete, then the lateness of the next considered y-job at Step 1 surpasses the current $L(\delta)$. If that y-job is a Type (b) y-job, then an instance of Alternative (b2) with that Type (b) y-job (abbreviated IA(b2)) is said to occur (the behavior alternatives were introduced in a wider context earlier in [26]). IA(b2) is dealt with in Phase 2, in which at least one Type (b) y-job is rescheduled within the interval of some earlier constructed bin.

Lemma 1. *If no IA(b2) at the first pass of Phase 1 occurs, i.e., the earliest arisen y-job y that could not have been included in schedule $S(B, y)$ is a Type (a) y-job), then a new kernel consisting of Type (a) y-jobs including job y occurs.*

Proof. First, note that all Type (a) y-jobs can be feasibly scheduled without surpassing the current allowable lateness and be completed within bin B. Hence, if the earliest arisen y-job y that could not have been included turns out to be a Type (a) y-job, a forced right-shift by some other y-job and/or the order change in the sequence of the corresponding Type (a) y-jobs must have occurred. Therefore, there must exist a corresponding delaying emerging job, and hence, a new kernel can be declared. □

If schedule $S(B, y)$ is not y-complete and no IA(b2) at the first pass of Phase 1 occurs, then Step 2 at Phase 1 is invoked. At that step, the new kernel from Lemma 3 is declared, and the current set of kernels and bins is respectively updated. Then, the first pass is called recursively for the newly formed partition; in particular, Step 1 is invoked for the earliest of the newly-arisen bins. This completes the description of the first pass.

Lemma 2. *If during the scheduling of bin B at the first pass, no IA(b2) arises, then this pass outputs a y-complete schedule $S(B, y)$ in which the lateness of no y-job is greater than $L(\delta)$.*

Proof. By Lemma 1 and the construction of the first pass, if at that pass, no IA(b2) arises, the constructed partial schedule $S(B, y)$ contains all the y-jobs with the lateness no more than $L(\delta)$, which proves the lemma. □

Phase 1, second pass (scheduling x-jobs): The second pass is invoked if no IA(b2) during the execution of the first pass occurs. The first pass will then output a y-complete schedule $S(B, y)$ (Lemma 2), which is the input for the second pass. At the second pass, the available x-jobs are included within the idle time intervals remaining in schedule $S(B, y)$. Now, we describe a modified version of a common Decreasing Next Fit heuristics, adapted to our problem for scheduling bin B (DNF heuristics for short). The heuristics deals with a list of the x-jobs for kernel K sorted in non-increasing order of their processing times, whereas the jobs with the same processing time are sorted in the non-decreasing order of their due dates (i.e., more urgent jobs appear earlier in that list). Iteratively, the DNF heuristics selects the next job x from the list and tries to schedule it at the earliest idle time moment t' at or behind its release time. If time moment $t' + p_i$ is greater than the allowable upper endpoint of the interval of bin B, job $x$is ignored, and the next x-job from the list is similarly considered (if no unconsidered x-job is left, then the second pass for bin B completes). Otherwise ($t' + p_i$ falls within the interval of bin B), the following steps are carried out.

(A) If job x can be scheduled at time moment t' without pushing an y-job included in the first pass (and be completed within the interval of the current bin), then it is scheduled at time t' (being removed from the list), and the next x-job from the list is similarly considered.

(B) Job x is included at time t' if it pushes no y-job from schedule $S(B, y)$. Otherwise, suppose job x, if included at time t', pushes a y-job, and let $S(B, +x)$ be the corresponding (auxiliary) partial ED-schedule obtained by rescheduling (right-shifting) respectively all the pushed y-jobs by the ED heuristics.

(B.1) If none of the right-shifted y-jobs in schedule $S(B, +x)$ result in a lateness greater than the current threshold $L(\delta)$ (all these jobs being completed within the interval of bin B), then job i is included, and the next x-job from the list is similarly considered.

We need the following lemma to complete the description of the second pass.

Lemma 3. *If at Step (B), the right-shifted y-job y results in a lateness greater than the threshold value $L(\delta)$, then there arises a new kernel K consisting of solely Type (a) y-jobs including job y.*

Proof. Since the lateness of job y surpasses $L(\delta)$, it forms part of a newly arisen kernel, the delaying emerging job of which is x. Furthermore (similarly as in Lemma 1), kernel K cannot contain any Type (b) y-job, as otherwise, such a y-job would have been included within the idle time interval in which job x is included. Hence, that Type (b) y-job could not succeed job x. □

(B.2) If the condition in Lemma 3 is satisfied, then the corresponding new kernel is declared, and the current set of kernels and bins is updated. Similarly, as in Step 2, the first pass is called for the first newly declared bin (initiated at time moment t') from the newly updated partition.

(B.3) If the lateness of some y-job surpasses the threshold $L(\delta)$ in schedule $S(B, +x)$, then the next x-job from the list is similarly processed from Step (A).

(B.4) If all the x-jobs from the list have been processed, then the second pass for scheduling bin B completes.

The following theorem tackles a frontier between the polynomially solvable instances and intractable ones for finding a Pareto-optimal frontier (the two related NP-hard problems are the scheduling problem with our primary objective and the derived bin packing problem):

Theorem 2. *Suppose, in the iteration of the binary search procedure with trial δ, all the bins are scheduled at Phase 1, i.e., no IA(b2) at Phase 1 arises. Then, a complete schedule in which the lateness of no job is greater than $L(\delta)$ is constructed in time $O(n^2 \log n)$. This schedule is Pareto sub-optimal, and it is Pareto optimal if no idle time interval in any bin is created.*

Proof. For every bin B, since no IA(b2) during its construction at Phase 1 arises, this phase will output a partial schedule for that bin, in which the lateness of no job is greater than $L(\delta)$ (Lemmas 2 and 3). At the same time, by our construction, no kernel job may result in a lateness greater than $L(\delta)$. Then, the maximum job lateness in the resultant complete schedule is no greater than $L(\delta)$. This schedule is Pareto sub-optimal due to the DNF heuristics (with a known sub-optimal approximation ratio). Furthermore, if no idle time interval in any bin is created, there may exist no feasible solution with the maximum job lateness $L(\delta)$ such that the latest scheduled job in it completes earlier than in the above generated solution; hence, it is Pareto optimal.

Let us now estimate the cost of Phase 1. For the purpose of this estimation, suppose n_1 (n_2, respectively) is the number of y-jobs (x-jobs, respectively). In the first pass, y-jobs are scheduled by the ED heuristics in time $O(n_1 \log n_1)$ if all the scheduled bins are y-complete. Otherwise, since no IA(b2) arises, each time the scheduling of a bin cannot be completed by all the y-jobs, a new kernel is declared that includes the corresponding y-jobs (Lemma 1). Since there may arise less than n_1 different kernels, the number of times a new kernel may arise is less than n_1. Thus, the same job will be rescheduled less than n_1 times (as a bin or a kernel job), which yields the time complexity of $O(n_1^2 \log n_1)$ for the first pass. Since the time complexity of the DNF heuristics is the same as that of ED heuristics used at the first pass and new kernels are similarly declared (Lemma 3), the cost of the second pass is similarly $O(n_2^2 \log n_2)$. Thus, the overall cost for the construction of all the bins is $O(n^2 \log n)$, which is also the cost of the overall scheme since the kernel jobs are scheduled by the ED heuristics. □

Note that the second condition in the above theorem is sufficient, but not necessary for minimizing the makespan; i.e., some bins in a feasible schedule respecting threshold $L(\delta)$ may have idle time intervals, but that schedule may minimize the makespan. In general, we rely on the efficiency of the heuristics used for packing our bins for the minimization of the makespan criterion. We also straightforwardly observe that by the DNF heuristicss, no unscheduled job may fit within any available gap in any bin without forcing some other job to surpass the current allowable lateness $L(\delta)$ at Phase 1, whereas the DNF-heuristicss tends to keep reserved "short" unscheduled jobs.

Continuing with the issue of the practical behavior, as we have mentioned in the Introduction, the ED heuristics turns out to be an efficient tool for a proper distribution of the jobs within the bins. In [9], computational experiments were conducted for 200 randomly generated instances of our problem. For the majority of these instances, the forced delay of kernel $K(S)$ in the initial ED-schedule S (i.e., the difference between the completion time of the corresponding delaying emerging job and $r(K)$) was neglectable. Therefore, the lateness of the corresponding overflow job was either optimal or very close to the optimal maximum job lateness (as no job of kernel K can be started earlier than at time $r(K)$ in any feasible solution). At the same time, since the ED heuristics creates no avoidable machine idle time interval, the minimum makespan is also achieved. Thus, for the majority of the tested problem instances, both objective criteria were simultaneously minimized or the maximum jobs lateness was very close to the optimum, whereas the makespan was optimal.

The computational experiments from [9] also yield that, for most of the tested instances, already for small trial δ's, the bins would have been scheduled in Phase 1 (i.e., no IA(b2) would arise). Moreover, there will be no avoidable gap left within the bins for these δ's. In other words, the conditions in Theorem 2 would be satisfied, and Phase 1 would successfully completed. In theory, however, we cannot guarantee such a behavior, and we proceed with our construction in Phase 2.

Phase 2: If the first condition in Theorem 2 is not satisfied, i.e., an IA(b2) at the first pass of Phase 1 occurs, then Phase 2 is invoked. Let y be the earliest arisen Type (b) y-job that could not have been included in schedule $S(B, y)$. In general, job j may be pushed by one or more other Type (b) y-jobs. Since the lateness of job y surpasses $L(\delta)$, a complete schedule respecting this threshold value cannot be created unless the lateness of job y is decreased. The next observation follows:

Observation 6. *Suppose an IA(b2) with (a Type (b) y-job) y at the first pass of Phase 1 in bin B arises. Then, the resultant maximum job lateness in bin B yielded by job y cannot be decreased unless job y and/or other Type (b) y-jobs currently pushing job y in bin B are rescheduled to some earlier bin(s).*

According to the above observation, in Phase 2, either job y is to be rescheduled to an earlier bin or a subset of the above Type (b) y-jobs (with an appropriate total processing time) is to be formed and all jobs from that subset are to be rescheduled to some earlier bin(s). In particular, if the lateness of job y surpasses $L(\delta)$ by amount λ, then the total processing time of jobs in such a subset is to be at least λ. By the construction, no new job can feasibly be introduced into any already scheduled bin B unless some jobs (with an appropriate total length) from that bin are rescheduled behind all the above Type (b) y-jobs. Such a job, which we call a substitution job for job y, should clearly be an emerging job for job y, i.e., $d_s > d_y$ (as otherwise, once rescheduled, its lateness will surpass $L(\delta)$).

Theorem 3. *If for an IA(b2) with a (Type (b) y-job) y, there exists no substitution job for job y, then there exists no feasible schedule respecting threshold value $L(\delta)$. Hence, there exists no Pareto-optimal solution with maximum job lateness $L(\delta)$.*

Proof. Due to Observation 6, job y or some other Type (b) y-job(s) pushing job y must be rescheduled to a bin preceding bin B. By the way we construct our bins, no new job can feasibly be introduced into any bin preceding bin B unless some (sufficiently long) job s from that bin is rescheduled behind all the above Type (b) y-jobs. $d_s < d_y$ must hold as otherwise job s or some job originally included before job y will surpass the current threshold; i.e., s is a substitution job. It follows that if there exists no substitution job, the lateness of at least one job from bin B or from its preceding bin will surpass threshold value $L(\delta)$, and hence, there may exist no feasible schedule respecting $L(\delta)$. The second claim immediately follows. □

Due to the above theorem, if these exists no substitution job, the call from the binary search procedure for the current δ halts with the failure outcome, and the procedure proceeds with the next (larger) trial value for δ; if there remains no such δ, the whole scheme halts.

The case when there exists a substitution job for job y is discussed now. We observe that there may exist a Pareto-optimal solution. To validate whether it exists, we first need to verify if there exists a feasible solution respecting threshold $L(\delta)$ (given that the call to the bin packing procedure was performed from the iteration in the binary search procedure with that threshold value). This task is easily verified if, for instance, there is a single substitution job s. Indeed, we may construct an auxiliary ED-schedule in which we activate that substitution job for job y (similarly as before, we increase artificially the release time of job s to that of job y, so that once the ED heuristics is newly applied to that modified problem instance, the substitution job s will be forced to be rescheduled behind job y). In the resultant ED-schedule with the activated job s, if the lateness of none of the rescheduled y-jobs is no greater than $L(\delta)$, Phase 1 can be resumed for bin B. Otherwise, there may exit no feasible solution respecting threshold $L(\delta)$ (hence, no Pareto-optimal solution with the maximum job lateness not exceeding $L(\delta)$ may exist), and the scheme can proceed with the next trial δ.

In practice, with a known upper limit on the total number of substitution jobs, the above activation procedure can be generalized to another procedure that enumerates all the possible subsets of substitution jobs in polynomial time. In general, if the number of substitution jobs is bounded only by n, the number of possible subsets of substitution jobs with total processing time λ or more might be

an exponent of n. In theory and practice, we may take one of the two alternative options: either we try all such possible non-dominated subsets building an implicit enumerative algorithm, or we may rather choose a heuristic approach generating a fixed number of "the most promising" subsets of substitution jobs in polynomial time. Based on our general framework, both implicit enumeration and heuristic algorithms might be implemented in different ways. The study of such algorithms, which would rely on the results of numerical experiments, is beyond the scope of this paper and could rather be a subject of independent research.

6. Conclusions

We have proposed a general framework for the solution of a strongly NP-hard bi-criteria scheduling problem on a single machine with the objectives to minimize the maximum job lateness and the makespan. Theorem 2 specifies explicitly the nature of the problem instances for which the Pareto optimal frontier can be found in polynomial time. Finding a feasible solution with the maximum job lateness not exceeding some constant threshold becomes intractable if the first condition in the theorem is not satisfied. Relying on the computational experiments from [9], we have observed that already for small trial δ's, the bins would have been scheduled in Phase 1, i.e., no IA(b2) would arise in Phase 1. In addition, since there will be no avoidable gap left within the bins for these δ's, the second condition in Theorem 2 would be satisfied, and Phase 1 would successfully complete in polynomial time. We have also observed that for the majority of the tested problem instances from [9], both objective criteria were simultaneously minimized or the maximum job lateness was very close to the optimum, whereas the makespan was optimal. Theoretically, we are unable to guarantee such a behavior, and hence, we described how the solution process can proceed in Phase 2. We have shown that it suffices to enumerate some subsets of the substitution jobs and generate the resultant feasible schedules. With a complete enumeration, we guarantee that one of these solutions will respect a given threshold, unless there exists no such a feasible schedule. In practice, if a complete enumeration of all dominant subsets turns out to take an admissible time, we may "switch" to heuristic methods that enumerate a constant number of the "most promising" subsets and the corresponding feasible solutions (In the latter case, theoretically, it is possible that we fail to create a feasible schedule respecting the threshold where such a feasible schedule exists. Then, the Pareto frontier that the algorithm will generate will miss the corresponding Pareto-optimal solution, i.e., it will not be "complete".).

By partitioning the scheduling horizon into kernel and bin intervals, we have reduced the problem of minimizing the makespan in a feasible schedule respecting a given threshold to that of minimizing the gaps in the bins in that schedule. Bin packing problems with this objective have been extensively studied in the literature, and a number of efficient enumeration algorithms and also heuristicss with good theoretical and also practical behavior exist. We showed how such a heuristic method can be adopted for packing our bins with jobs with release times and due dates (the latter job parameters impose additional restrictions, not present in bin packing problems). In this way, our framework has reduced the bi-criteria scheduling problem to the above two single-criterion problems, inducing four basic types of algorithms for the bi-criteria problem (depending on whether an exact or heuristic method is incorporated for either of the two problems). For the future work, it would be interesting to test particular implementations of these algorithms and to verify their performance for problem instances that significantly differ from those tested in [9].

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

References

1. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP—Completeness*; W. H. Freeman and Company: San Francisco, CA, USA, 1979.
2. Jackson, J.R. Schedulig a Production Line to Minimize the Maximum Tardiness. In *Manegement Scince Research Project, University of California*; Office of Technical Services: Los Angeles, CA, USA, 1955.

3. Schrage, L. Obtaining Optimal Solutions to Resource Constrained Network Scheduling Problems. Unpublished work, 1971.
4. Potts, C.N. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Oper. Res.* **1980**, *28*, 1436–1441. [CrossRef]
5. Hall, L.A.; Shmoys, D.B. Jackson's rule for single-machine scheduling: Making a good heuristic better. *Mathem. Oper. Res.* **1992**, *17*, 22–35. [CrossRef]
6. Nowicki, E.; Smutnicki, C. An approximation algorithm for single-machine scheduling with release times and delivery times. *Discret Appl. Math.* **1994**, *48*, 69–79. [CrossRef]
7. Larson, R.E.; Dessouky, M.I. heuristic procedures for the single machine problem to minimize maximum lateness. *AIIE Trans.* **1978**, *10*, 176–183. [CrossRef]
8. Kise, H.; Ibaraki, T.; Mine, H. Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times. *J. Oper. Res. Soc. Japan* **1979**, *22*, 205–223. [CrossRef]
9. Vakhania, N.; Perez, D.; Carballo, L. Theoretical Expectation versus Practical Performance of Jackson's heuristic. *Math. Probl. Eng.* **2015**, *2015*, 484671. [CrossRef]
10. McMahon, G.; Florian, M. On scheduling with ready times and due dates to minimize maximum lateness. *Oper. Res.* **1975**, *23*, 475–482. [CrossRef]
11. Carlier, J. The one–machine sequencing problem. *Eur. J. Oper. Res.* **1982**, *11*, 42–47. [CrossRef]
12. Sadykov, R.; Lazarev, A. Experimental comparison of branch-and-bound algorithms for the $1|r_j|L_{max}$ problem. In Proceedings of the Seventh International Workshop MAPSP'05, Siena, Italy, June 6–10 2005; pp. 239–241.
13. Grabowski, J.; Nowicki, E.; Zdrzalka, S. A block approach for single-machine scheduling with release dates and due dates. *Eur. J. Oper. Res.* **1986**, *26*, 278–285. [CrossRef]
14. Larson, M.I. Dessouky and Richard E. DeVor. A Forward-Backward Procedure for the Single Machine Problem to Minimize Maximum Lateness. *IIE Trans.* **1985**, *17*, 252–260, doi:10.1080/07408178508975300. [CrossRef]
15. Pan, Y.; Shi, L. Branch-and-bound algorithms for solving hard instances of the one-machine sequencing problem. *Eur. J. Oper. Res.* **2006**, *168*, 1030–1039. [CrossRef]
16. Liu, Z. Single machine scheduling to minimize maximum lateness subject to release dates and precedence constraints. *Comput. Oper. Res.* **2010**, *37*, 1537–1543. [CrossRef]
17. Gharbi, A.; Labidi, M. Jackson's Semi-Preemptive Scheduling on a Single Machine. *Comput. Oper. Res.* **2010**, *37*, 2082–2088, doi:10.1016/j.cor.2010.02.008. [CrossRef]
18. Lenstra, J.K.; Rinnooy Kan, A.H.G.; Brucker, P. Complexity of machine scheduling problems. *Ann. Discret. Math.* **1977**, *1*, 343–362.
19. Kacem, I.; Kellerer, H. Approximation algorithms for no idle time scheduling on a single machine with release times and delivery times. *Discret. Appl. Math.* **2011**, *164*, 154–160, doi:10.1016/j.dam.2011.07.005. [CrossRef]
20. Lazarev, A. The Pareto-optimal set of the NP-hard problem of minimization of the maximum lateness for a single machine. *J. Comput. Syst. Sci. Int.* **2006**, *45*, 943–949. [CrossRef]
21. Lazarev, A.; Arkhipov, D.; Werner, F. Scheduling Jobs with Equal Processing Times on a Single Machine: Minimizing Maximum Lateness and Makespan. *Optim. Lett.* **2016**, *11*, 165–177. [CrossRef]
22. Carlier, J.; Hermes, F.; Moukrim, A.; Ghedira, K. Exact resolution of the one-machine sequencing problem with no machine idle time. *Comp. Ind. Eng.* **2010**, *59*, 193–199, doi:10.1016/j.cie.2010.03.007. [CrossRef]
23. Chrétienne, P. On single-machine scheduling without intermediate delays. *Discrete Appl. Math.* **2008**, *156*, 2543–2550. [CrossRef]
24. Ehrgott, M. *Multicriteria Optimization*; Springer Science & Business Media: Berlin, Germany, 2005; Volume 491.
25. Chinos, E.; Vakhania, N. Adjusting scheduling model with release and due dates in production planning. *Cogent Eng.* **2017**, *4*, 1–23. [CrossRef]
26. Vakhania, N. A better algorithm for sequencing with release and delivery times on identical processors. *J. Algorithms* **2003**, *48*, 273–293. [CrossRef]