PRODUCTION & MANUFACTURING | RESEARCH ARTICLE

Adjusting scheduling model with release and due dates in production planning†

Elisa Chinos and Nodari Vakhania

Cogent Engineering (2017), 4: 1321175
PRODUCTION & MANUFACTURING | RESEARCH ARTICLE

Adjusting scheduling model with release and due dates in production planning†

Elisa Chinos¹ and Nodari Vakhania¹*

Abstract: Motivated by the conjecture that an interaction between scheduling and pre-scheduling phases in production planning may give certain benefits, we conduct a detailed study of the optimality conditions and dominance relations for a strongly NP-hard single-machine scheduling model when jobs have release and due-dates and the objective is to minimize maximum job lateness. By exploring the inherent structure of the problem, we establish the optimality conditions when the problem can be efficiently solved. We come to an NP-hard special case of the problem with only two possible job release times, that as we show allows stricter dominance rules and optimality conditions verifiable in polynomial time. The established properties give a potential of a beneficial interaction between scheduling and pre-scheduling phases in production planning, and also provide basic theoretical background for the construction of efficient heuristic and implicit enumerative algorithms.

Subjects: Applied Mathematics; Computer Mathematics; Mathematical Modeling; Foundations & Theorems

Keywords: production planning; scheduling model; scheduling algorithm; heuristic; release time; delivery time; due-date

1. Introduction

The term scheduling refers to the assignment of a set of requests to the given set of resources over time with the objective to optimize a given objective criterion. The requests are called jobs or tasks and a resources are called machines or processors, whereas the aim is to choose the order of

ABOUT THE AUTHOR

Nodari Vakhania is a titular professor at the Centro de Investigación en Ciencias, the State University of Morelos, Mexico and at the Institute of Computational Mathematics of the Georgian Academy of Sciences. He has received his PhD degree in mathematical cybernetics at the Russian Academy of Sciences in 1991, and the doctoral degree (habilitation) in mathematical cybernetics at the Georgian Academy of Sciences in 2004. His main research interests include design and analysis of algorithms, discrete optimization, computational complexity and scheduling theory. He has been interested in scheduling problems with job release times and due dates, and has developed novel efficient methods for their solution (so-called Blesscmore methods). The current work, carried out in collaboration with his PhD student Elisa Chinos, is part of this project.

PUBLIC INTEREST STATEMENT

Scheduling problems are mathematical problems formalized for the solution of diverse practical real-life problems. They deal with a finite set of requests or jobs to be performed (or scheduled) on a finite (and limited) set of resources called machines or processors. The aim is to choose the order of processing the jobs on the machines so as to meet a given objective criterion. In this paper we study a scenario with a single machine when jobs have arrival and due dates: a job cannot be assigned to the machine before its arrival time, whereas it is desirable to complete it by its due date; if this is not possible, we wish to minimize the maximum job deviation from its due date (the so-called lateness). The problem is known to be intractable. Still, we derive conditions that lead to its optimal solution in polynomial time.
processing the jobs on the machines so as to meet a given objective criteria. Therefore, a scheduling problem is characterized by the three distinct components: the tasks or activities that one wishes to perform, the resources available for their implementation, and the aims or objectives that one wants to achieve (that identify the best solutions). Different characteristics of jobs and machines together with different optimality criteria originate a vast amount of the scheduling problems.

Scheduling is an important phase in production planning, that, in turn, constitutes part of a wider production cycle. Once a product is designated, its production process needs to be determined. Decisions that are taken at the earlier phases of production planning before the scheduling phase (such as allocation and distribution of resources, raw materials, time limits, etc.), determine a particular scheduling model that is to be solved, i.e. this scheduling model essentially depends on the decisions made at the earlier (pre-scheduling) stages of production planning. In practice, there is a little interaction between the scheduling stage and the earlier stages in production planning, in the sense that the earlier made decisions do not normally change depending on the (potential) outcomes at the scheduling stage, i.e. the scheduling model itself doesn’t change. However, such an interaction might be beneficial since two similar scheduling models may have drastically different complexity status and hence admit solution methods of different efficiency. Based on the simulation results for the originally selected scheduling model, the decisions taken at the earlier (pre-scheduling) stages may potentially be adjusted, which, in turn, may adjust the scheduling model itself. This is often difficult since the most of the arisen scheduling models turn out to be computationally intractable (NP-hard in formal terms), and hence the interaction becomes complicated because of a considerable amount of time needed to solve the corresponding scheduling problem for delivering a solution with the desired quality. Whereas a general solution method for the initially selected scheduling model may require inadmissible computational time, the instances of that model that occur in a given application may have specific characteristics or can be adjusted to satisfy some particular conditions and properties. To this end, the study of the inherent structure of the initially selected scheduling model and the optimality conditions for that model, i.e. the conditions when the model can be efficiently solved, is important.

In this paper, we carry out this kind of study for a single-machine scheduling model with \( n \) requests or jobs, where every request is released at an integer time moment, has a processing requirement on the machine, and a due-date that specifies a desirable time moment for its completion. A job completed behind its due-date is said to be late. The objective is to minimize the maximum lateness of any job, i.e. the difference between its completion time and due-date.

According to the conventional three-field notation introduced by Graham, Lawler, Lenstra, and Rinnooy Kan (1979) the problem is abbreviated as \( 1|r_j|L_{\text{max}} \): the first field indicates the machine (single-processor) environment, the second field specifies job parameters (\( r_j \) stands for the release time of job \( j \)), and in the third field the objective criterion is given (here \( L_{\text{max}} \) stands for the maximum job lateness, in the second field the processing time is omitted since it is an obligatory parameter for any scheduling problem). There is an equivalent formulation of the problem in which job due-dates are replaced by the delivery times, abbreviated \( 1|r_j, q_j|C_{\text{max}} \) (\( q_j \) stands for the delivery times): once job completes on the machine, it needs to be delivered to the customer. Since the delivery is accomplished by an independent unit, it requires no machine time. The objective is then to minimize the maximum job full completion time that includes its delivery.

In Section 2 we show why these two versions are equivalent and give their detailed descriptions. Because of the equivalence, we shall refer to any of the two versions interchangeably. The problem is known to be strongly NP-hard (Garey & Johnson, 1979). There are exact implicit enumerative algorithms, see for instance, McMahon and Florian (1975) and Carlier (1982). An efficient heuristic method that is commonly used for problem \( 1|r_j|L_{\text{max}} \) was proposed long time ago by Jackson (1955) for the version of the problem without release times, and then was extended by Schrage (1971) for taking into account job release times. The extended Jackson’s heuristic (J-heuristic, for short) iteratively, at
each scheduling time $t$ (given by job release or completion time), among the jobs released by time $t$ schedules one with the the largest delivery time (or smallest due-date). Since the number of scheduling times is $O(n)$ and at each scheduling time search for a minimal/maximal element in an ordered list is accomplished, the time complexity of the heuristic is $O(n \log n)$. J-heuristic is sometimes referred to as EDD-heuristic (Earliest Due-Date) or alternatively, LDT-heuristic (Largest Delivery Time).

J-heuristic gives the worst-case approximation ratio of 2, i.e. it delivers a solution which is at most twice worse than an optimal one. There are polynomial time algorithms with a better approximation for problem $1|r_j, q_j|C_{\text{max}}$. Potts (1980) has proposed a modification of J-heuristic that repeatedly applies J-heuristic $O(n)$ times and obtains an improved approximation ratio of 3/2. Hall and Shmoys (1992) have expanded the later algorithm resulting in an improved approximation ratio of 4/3 by applying J-heuristic to the forward and backward versions of the problem simultaneously, and also proposed two polynomial approximation schemes (their algorithms allow partial order on the set of jobs). The idea of the usefulness of the simultaneous application of J-heuristic in forward and backward fashions was earlier exploited by Nowicki and Smutnicki (1994). In a more recent work (Vakhania Perez, & Carballo, 2013), the magnitude $\kappa$, no-more than the optimal objective value divided by the specially determined job processing time, is used to derive a more accurate worst-case approximation ratio of $1 + 1/\kappa$ for J-heuristic. The value of parameter $\kappa$ can be brutally determined in time $O(n \log n)$ using an easily calculable lower bound on the optimal objective value instead of using the optimal objective value itself.

As to the special cases of our problem, if all job release times or delivery times are equal, J-heuristic (its original version) gives an optimal solution, and with integer release times and unit processing times, it also delivers an optimal solution. If job release times, processing times and delivery time are restricted in such way that each $r_j$ lies in the interval $[q - q_j, p_j - A, q - q_j - A]$, for some constant $A$ and suitably large $q$, then the problem can also be solved in time $O(n \log n)$, see Hoogeveen (1995). Garey, Johnson, Simons, and Tarjan (1981) have proposed an $O(n \log n)$ algorithm for the feasibility version with equal-length jobs (in the feasibility version job due-dates are replaced by deadlines and a schedule in which all jobs complete by their deadlines is looked for). Later in Vakhania (2004) was proposed an $O(n^2 \log n)$ algorithm for the minimization version with two possible job processing times.

For other related criteria, in Vakhania (2009) an $O(n^3 \log n)$ algorithm that minimizes the number of late jobs with release times on a single-machine when job preemptions are allowed. Without preemptions, two polynomial-time algorithms for equal-length jobs on single machine and on a group of identical machines were proposed in Vakhania (2013, 2012), respectively, with time complexities $O(n^2 \log n)$ and $O(n^3 \log n \log p_{\max})$, respectively.

Problem $1|r_j, q_j|C_{\text{max}}$ has been shown to be useful for the solution of the multiprocessor scheduling problems. For example, for the feasibility version with $m$ identical machines and equal-length jobs, algorithms with the time complexities $O(n^3 \log \log n)$ and $O(n^3 m)$ were proposed in Simons (1983) and Simons and Warmuth (1989), respectively. In Vakhania (2003) was proposed an $O(q_{\max} mn \log n + O(mn))$ algorithm for the minimization version of the latter problem, where $q_{\max}$ is the maximal job delivery time and $n < n$ is a parameter.

The problem $1|r_j, q_j|C_{\text{max}}$ is commonly used for the solution of more complex job-shop scheduling problems as well. A solution of the former problem gives a strong lower bound for job-shop scheduling problems. In the classical job-shop scheduling problem the preemptive version of J-heuristic applied for a specially derived single-machine problem immediately gives a lower bound, see, for example, Carlier (1982), Carlier and Pinson (1989) and Brinkkotter and Brucker (2001) and more recent works of Gharbi and Labidi (2010) and Della Croce and T’kindt (2010). Carlier and Pinson (1998) have used a straightforwardly generalized J-heuristic for the solution of the multiprocessor job-shop problem with identical machines. It can also be adopted for the case when parallel machines are unrelated, see Vakhania and Shchepin (2002). J-heuristic can be useful for parallelizing the
computations in scheduling job-shop Perregaard and Clausen (1998), and also for the parallel batch scheduling problems with release times Condotta, Knust, and Shakhlevich (2010).

Besides the above mentioned applications in multiprocessor, shop and batch scheduling problems, our problem has numerous immediate real-life applications in various production chains, CPU time sharing in operating systems (jobs being the processes to be executed by the processor), wireless sensor network transmission range distribution (where jobs are mobile devices with their corresponding ranges that can be modeled as release and due dates).

By exploring the inherent structural properties of the problem, here we derive new optimality conditions when practical special cases of our problem can be solved optimally in a low degree polynomial time. In particular, we provide explicit conditions that lead to an efficient solution of the problem by a mere application of J-heuristic. Then we study further useful structural properties of the J-schedules (ones created by J-heuristic) leading to the optimal solution of other versions of the problem with the same computational cost as that of J-heuristic. Finally, we focus on a special case of our problem with only two allowable job release times \(r_1\) and \(r_2\) with \(r_1 < r_2\) (abbreviated \(1|\{r_1, r_2\}|\mathit{max}\) or \(1|\{r_1, r_2\}, \{q_j\}|\mathit{max}\)). Although the latter problem remains NP-hard, it admits stricter dominance rules and optimality conditions leading to the corresponding polynomial-time verification procedures (and the reduction of the search space).

Employing the presented optimality and dominance conditions, a practitioner may simulate the scheduling phase under these conditions in real time (solving efficiently the scheduling problem under these conditions). Then it is verified whether the obtained solution also solves the original instance, i.e. if it is feasible for that instance. Otherwise, it might be possible to adopt the production process so that the optimality conditions are met. Carrying out this kind of simulation for a variety of arisen in a given production instances, a practitioner may adjust the pre-scheduling decisions and hence convert the initially determined scheduling model to another more restricted one admitting an efficient solution method. In this way, the production process might be adjusted to the optimality and dominance conditions resulting in a useful interaction between pre-scheduling and scheduling phases in production planning.

In the next subsection we give the basic concepts and some other preliminaries. In Section 3 we study efficiently identifiable conditions and cases when our generic problem can be solved in polynomial time. In Section 4 we show that the special case of the problem with only two allowable job due-dates remains NP-hard. Section 5 is devoted to the latter problem. In the first part of this section we give general properties for the polynomial time solution, whereas in Sections 5.1 and 5.2 we derive stricter dominance relations and other particular versions that can be efficiently solved. We give the final remarks in Section 6.

2. Basic concepts and definitions

Now we give a formal definition of our general scheduling problem. We have \(n\) jobs \(j (j = 1, \ldots, n)\) and single machine available at the time 0. Each job \(j\) become available at its release time or head \(r_j\), needs continuous processing time \(p_j\) on the machine, and needs an additional delivery time or tail \(q_j\) after the completion on the machine (note that \(q_j\) needs no machine time in our model). The heads and tails are non-negative integral numbers while a processing time is a positive integer. A feasible schedule \(S\) assigns to each job \(j\) a 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 only one job at the time. The completion time of job \(j\), \(C_j(S) = t_j(S) + p_j\); the full completion time of job \(j\) in \(S\), \(C_j(S) = t_j(S) + q_j\). Our objective is to find an optimal schedule, i.e. a feasible schedule \(S\) with the minimal value of the maximal full job completion time \(|S| = \max_j C_j\) (called the makespan).
In an equivalent formulation $1|r_j|L_{\text{max}}$, the delivery time $q_j$ of every job $j$ is replaced by the due-date $d_j$, which is the desirable time for the completion of job $j$. Here the maximum job lateness $L_{\text{max}}$ (the difference between the job completion time and its due-date) needs to be minimized.

Given an instance of $1|r_j, q_j|C_{\text{max}}$, one can obtain an equivalent instance of $1|r_j|L_{\text{max}}$ as follows. Take a suitably large constant $K$ (no less than the maximum job delivery time) and define due-date of every job $j$ as $d_j = K - q_j$. Vice-versa, given an instance of $1|r_j|L_{\text{max}}$, an equivalent instance of $1|r_j, q_j|C_{\text{max}}$ can be obtained by defining job delivery times as $q_j = D - d_j$, where $D$ is a suitably large constant (no less than the maximum job due date). It is straightforward to see that the pair of instances defined in this way are equivalent; i.e. whenever the makespan for the version $1|r_j, q_j|C_{\text{max}}$ is minimized, the maximum job lateness in $1|r_j|L_{\text{max}}$ is minimized, and vice-versa. We refer the reader to an early paper by Bratley, Florian, and Robillard (1973) for more details.

Next, we give a more detailed description of J-heuristic. It 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 (the maximum delivery time, alternatively) is assigned to the machine (ties being broken by selecting a longest job). Iteratively, the next scheduling time is either the completion time of the latest assigned so far job to the machine or the minimum release time of a yet unassigned job, whichever is more (as no job can be started before the machine gets idle neither before its release time). And again, among all jobs released by this scheduling time a job with the minimum due-date (the maximum delivery time, alternatively) is assigned to the machine. Note that the heuristic creates no gap that can be avoided always scheduling an already released job once the machine becomes idle, whereas among yet unscheduled jobs released by each scheduling time it gives the priority to a most urgent one (i.e. one with the smallest due-date or alternatively the largest delivery time).

We will use $\sigma$ for the initial J-schedule, i.e. one obtained by the application of Jackson’s heuristic (J-heuristic, for short) to the originally given problem instance, and $S_{\text{opt}}$ for an optimal schedule.

A J-schedule has a particular structure that enables us to deduce our conditions. Two basic classes of jobs in a J-schedule might be distinguished: the “urgent” (critical) and “non-urgent” (non-critical) ones. Critical jobs (that we call kernel jobs) are ones which may potentially realize the maximum value of the objective function in an optimal schedule $S_{\text{opt}}$ (we call such jobs overflow jobs). In a J-schedule, the critical jobs form tight sequences that extend to limited time intervals. We call such sequences of critical jobs kernels. Then the non-critical jobs (that we call emerging jobs) are to be distributed within the remained intervals (in between the kernels) in some optimal fashion. Starting from Section 3, we derive conditions when this can be done in polynomial time. We need to introduce some preliminary definitions before we define formally the above mentioned notions.

A J-schedule may contain a gap, which is its maximal consecutive time interval in which the machine is idle. We assume that there occurs a 0-length gap $(c_j, t_j)$ whenever job $i$ starts at its earliest possible starting time, i.e. its release time, immediately after the completion of job $j$; here $t_j$ ($c_j$, respectively) denotes the starting (completion, respectively) time of job $j$.

A block in a J-schedule is its consecutive part consisting of the successively scheduled jobs without any gap in between preceded and succeeded by a (possibly a 0-length) gap. J-schedules have useful structural properties. The following basic definitions, taken from Vakhania (2004), will help us to expose these properties.

Among all jobs in a J-schedule $S$, we distinguish the ones which full completion time realizes the maximum full completion time in $S$, the latest scheduled such job is called the overflow job in $S$. We denote the overflow job in $S$ by $a(S)$. The block critical of $S$, $B(S)$ is the block containing $a(S)$.
Job $e$ is called an emerging job in schedule $S$ if $e \in B(S)$ and $a_e < a_{o(S)}$. The latest scheduled emerging job before the overflow job $o(S)$ is called live and is denoted by $l$.

The kernel of $S$, $K(S)$, is a sequence consisting of the jobs scheduled in schedule $S$ between the live emerging job $l(S)$ and the overflow job $o(S)$, not including $l(S)$ but including $o(S)$. Abusing slightly the terminology, we shall also use kernel for the corresponding set of jobs, and denote by $r(K)$ the minimum job release time in kernel $K$.

It follows that every kernel is contained in some block in $S$, and the number of kernels in $S$ equals to the number of the overflow jobs in it. Furthermore, since any kernel belongs to a single block, it may contain no gap. Note that if a J-schedule $S$ has no emerging job then it will also have no kernel; then the overflow job in schedule $S$ will not be part of any kernel. In general, we shall use $E(S)$ for the set of all emerging jobs scheduled before kernel $K(S)$ in schedule $S$.

Figure 1 illustrates the above introduced notions for an instance with 6 jobs below. Schedule $\sigma$ consists of two blocks $B_1$ and $B_2$. Block $B_2$ is critical containing the overflow job 5 and kernel $K(\sigma) = \{6, 5\}$. Job 4 is the only emerging job. There is a gap $[16, 18]$ in $\sigma$.

If a J-schedule $S$ is not optimal then there must exist an emerging job forcing the delay of the kernel jobs in $K(S)$ and that of the overflow job $o(S)$ which must be restarted earlier (see Lemma 2 a bit later). We denote the amount of this forced delay by $\Delta_l = c_l(S) - r(K(S))$.

**Observation 1** In any feasible schedule, the jobs of kernel $K(\sigma)$ may be restarted earlier by at most $\Delta_l$ time units.

**Proof** Immediately follows from the definition of $\Delta_l$ and from the fact that no job of kernel $K(\sigma)$ is released earlier than at time $r(K(\sigma))$. \qed

In order to reduce the maximum job lateness ($\mathbb{L}_\infty(S)$) in $S$, we apply an emerging job $e$ for $K(S)$, that is, we reschedule $e$ after $K(S)$. Technically, we accomplish this into two steps: first we increase artificially the release time of $e$, assigning to it a magnitude, no less than the latest job release time in $K(S)$;
then we apply the J-heuristic to the modified in this way problem instance. Since now the release time of e is no less than that of any job of K(S), and \( q_e \) is less than any job tail from K(S), the J-heuristic will give priority to the jobs of K(S) and job e will be scheduled after all these jobs.

If we apply the live emerging job \( l \), then it is easy to see that there will arise a gap before the jobs of kernel K(S) if no other emerging job scheduled in S behind kernel K(S) gets included before kernel K(S) taking the (earlier) position of job \( l \). In general, while we apply any emerging job \( e \in E(S) \), we avoid such a scenario by increasing artificially the release time of any such an emerging job which may again push the jobs in kernel K(S) in the newly constructed schedule that we call a complementary to S schedule and denote by \( S_e \).

Below we illustrate the construction of schedule \( \sigma_e \). The table specifies a problem instance with 9 jobs. Schedules \( \sigma \) and \( \sigma_e \) are illustrated in Figures 2 and 3, respectively (the numbers within the circles stand for the full completion times of the corresponding jobs).

**Figure 2. Schedule \( \sigma \).**

- \( r_1 = 0 \)
- \( p_1 = 5 \)
- \( q_1 = 30 \)
- \( r_2 = 0 \)
- \( p_2 = 4 \)
- \( q_2 = 25 \)
- \( r_3 = 0 \)
- \( p_3 = 4 \)
- \( q_3 = 21 \)
- \( r_4 = 0 \)
- \( p_4 = 2 \)
- \( q_4 = 17 \)
- \( r_5 = 0 \)
- \( p_5 = 10 \)
- \( q_5 = 10 \)
- \( r_6 = 20 \)
- \( p_6 = 7 \)
- \( q_6 = 30 \)
- \( r_7 = 20 \)
- \( p_7 = 8 \)
- \( q_7 = 28 \)
- \( r_8 = 20 \)
- \( p_8 = 5 \)
- \( q_8 = 22 \)
- \( r_9 = 20 \)
- \( p_9 = 4 \)
- \( q_9 = 16 \)

**Figure 3. Schedule \( \sigma_e \).**
3. The cases and conditions for polynomial time solution of the problem

In this section we give conditions leading to the efficient solution of our generic problem. These conditions are derived as a consequence of the analysis of J-schedule \( \sigma \) immediately and hence provide an \( O(n \log n) \) time decision.

**Lemma 1** If the overflow job \( o(\sigma) \) is scheduled at its release time, then \( \sigma \) is optimal.

**Proof** If \( o(\sigma) \) is scheduled at its release time \( r_{o(\sigma)} \), then its starting time \( t_{o(\sigma)} = r_{o(\sigma)} \). Hence job \( o(\sigma) \) starts at its early starting time in \( \sigma \) and ends at its minimum completion time. Therefore, \( o(\sigma) \) can not be rescheduled earlier in \( \sigma \), i.e. there is no scheduling \( \sigma' \) such that \( |\sigma'| < |\sigma| \) and \( \sigma \) is optimal.

If \( o(\sigma) \) is not scheduled at its release time, then the starting time of \( o(\sigma) \) must be the completion time of some other job. Another case when schedule \( \sigma \) is optimal, is when the tails of all the jobs scheduled in the critical block before \( o(\sigma) \) are greater than \( o(\sigma) \):

**Lemma 2** If the critical block \( B(\sigma) \) does not contain an emerging job (equivalently, it has no kernel), then schedule \( \sigma \) is optimal.

**Proof** Assume that \( \sigma \) contains a critical block \( B(\sigma) \) which does not contain an emerging job, i.e. for all \( e \in B(\sigma) \) scheduled before job \( o(\sigma) \), \( q_e \geq q_{o(\sigma)} \).

Suppose there is a non-empty sequence \( E \) of jobs in \( B(\sigma) \) scheduled before \( o(\sigma) \). If we reschedule some jobs of \( E \) after \( o(\sigma) \) the in the resultant schedule \( \sigma' \) at least one job \( e \) will be completed no earlier than at the moment \( C_{o(\sigma)}(\sigma) \). But as \( e \) is not an emerging job, \( q_e \geq q_{o(\sigma)} \) and therefore the full completion time of \( e \), \( C_e(\sigma) \), will be no less than \( C_{o(\sigma)}(\sigma) \). Then \( |\sigma'| > |\sigma| \). Therefore, \( \sigma \) is optimal.

If set \( E \) is empty, then \( o(\sigma) \) is scheduled in \( \sigma \) at its release time, therefore by Lemma 1, \( \sigma \) is optimal.

The case when \( \sigma \) has no kernel is quite similar.

The following two lemmas state earlier known results that we have mentioned in the introduction. Their proofs are presented for the sake of completeness of this presentation.

**Lemma 3** If all release times \( r_i (i = 1, \ldots, n) \) of jobs in \( o \) are equal to a constant \( r \), then \( \sigma \) is optimal.

**Proof** As all jobs \( i(i = 1, \ldots, n) \) are released at the time \( r \), then the J-heuristic in each iteration schedules the job \( j \) with largest tail. This gives us a schedule \( \sigma \), such that jobs are scheduled in non-increasing order. Then \( \sigma \) cannot contain an emerging job and by Lemma 2, \( \sigma \) is optimal.

**Lemma 4** Let \( \sigma \) be a J-schedule with jobs \( i(i = 1, \ldots, n) \) with integer release times \( r_i \) and unit processing times \( p_i = 1 (i = 1, \ldots, n) \). Then \( \sigma \) is optimal.

**Proof** Since the release time \( r_i \) each job \( i \) is an integer number and its processing time is 1, during the execution of that job no other more urgent job may be released. Hence, at each scheduling time \( t \), J-heuristic, among all the released by that time moment jobs, will include one with the largest tail.

In addition, in \( \sigma \), every job is scheduled at its release time \( r_i \) or at the completion time of another job. If job \( o(\sigma) \) is scheduled at time \( t = r_{o(\sigma)} \) then \( \sigma \) is optimal by Lemma 1. If \( o(\sigma) \) is scheduled at the completion time of another job \( k \) so that job \( o(\sigma) \) was released before the completion time of that job, then \( q_k \geq q_{o(\sigma)} \). Hence, there may exist no emerging job and \( \sigma \) is optimal by Lemma 2.

**Lemma 5** Let jobs in \( J = \{j_1, \ldots, j_n\} \) be ordered by a non-decreasing sequence of their release times. Then \( \sigma \) is optimal if for any neighboring jobs \( j_i \) and \( j_{i+1} (j = 1, \ldots, n-1) \), \( r_{j_{i+1}} \geq r_j + p_j \).
By the condition in the lemma, every job $i_j \ (j = 1, \ldots, n)$ is scheduled in $\sigma$ in the interval $[r_{i_j}, r_{i_j} + p_{i_j}]$. So, the starting time of each job in $\sigma$ is $t_{i_j} = r_{i_j}$, i.e. each job $i_j$ begins at its early starting time in $\sigma$. This is true, in particular, for job $o(\sigma)$ and therefore, by Lemma 1, $\sigma$ is optimal. $\square$

Figure 4 illustrates schedule $\sigma$ for an instance possessing the above indicated property.

For the given $k$ job release times, $r_1 \leq \cdots \leq r_k$, let $J_i (i = 1, \ldots, k)$ be the set of jobs released at the time $r_i$.

**Theorem 1** For a given instance of problem $1|r_j, q_j|C_{\text{max}}$ the initial $J$-schedule $\sigma$ is optimal if $o(\sigma) \in J_1$.

**Proof** By the condition, at any scheduling time behind release times $r_2, \ldots, r_k$, job $o(\sigma)$ was released. Then by $J$-heuristic, for each job $i$ scheduled in $\sigma$ before job $o(\sigma)$, $q_i \geq q_{o(\sigma)}$. Hence, $\sigma$ does not contain an emerging job and it is optimal by Lemma 2. $\square$

Assume $K$ is any kernel which forms part of some $J$-schedule $S$ ($K$ can be a kernel of other $J$-schedule distinct from $S$). Then we will say that job $e \in E(S)$ is scheduled within kernel $K$ if there is at least one job from that kernel scheduled before and after job $e$ in schedule $S$.

**Lemma 6** If job $e$ is scheduled within kernel $K(\sigma)$ in schedule $S$ then $|S| > |\sigma|$.

**Proof** First, it is easy to see that no job from kernel $K(\sigma)$ scheduled before job $e$ can be the overflow job in schedule $S$. Neither job $e$ can be the overflow job in $S$ as it is succeeded by at least one kernel job $j \in K(\sigma)$ with $q_j \geq q_e$. Furthermore, no job $k$ scheduled after kernel $K(\sigma)$ can be the overflow job in schedule $S$ as the right-shift of such a job in schedule $S$ cannot be more than that of a more urgent kernel job.
It follows that only a job from kernel $K(\sigma)$ scheduled after job $e$ may be the overflow job in schedule $S$. But because of the forced right-shift imposed by job $e$, the job $j$ from kernel $K(\sigma)$ scheduled the last in schedule $S$ cannot complete earlier in schedule $S$ than job $a(\sigma)$ was completed in schedule $\sigma$, i.e. $c_j(S) \geq c_{\text{overflow}}(\sigma)$. At the same time, by the definition of kernel $K(\sigma)$ job $j$ is no less urgent than job $a(\sigma)$, i.e. $q_j \geq q_{\text{binary}}$. Then $c_j(S) \geq C_{\text{overflow}}(\sigma)$ and hence $|S| \geq |\sigma|$. □

As a consequence of Lemmas 2 and 6 we get the following corollary:

**Corollary 1**  In any schedule better than $\sigma$, an emerging job is rescheduled behind kernel $K(\sigma)$.

**4. The NP-hardness of** $1||r_1, r_2, q_1, q_2||C_{max}$

From here on, we shall focus our attention on the special case of our problem with only 2 allowable job release times and tails. As we have seen earlier, $J$-heuristic delivers an optimal schedule whenever we have a single release time or a single tail (or due-date), i.e. the problems $1|q_j|C_{max}$ and $1|r_j|C_{max}$ are solvable in an almost linear time $n \log n$ (the general setting $1|r_j, q_j|C_{max}$ being strongly NP-hard). Now we show that we arrive at an NP-hard problem by allowing two distinct release times or tails; i.e. the problem $1|\{r_1, r_2\}, \{q_1, q_2\}|C_{max}$ is already NP-hard. For the convenience, let us consider the version with due-dates $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{max}$.

**Theorem 2**  $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{max}$ is NP-hard.

**Proof** We use the reduction from an NP-hard SUBSET SUM problem for the feasibility version of $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{max}$ in which we wish to know if there exists a feasible schedule that meets all job due-dates (i.e. in which all jobs are completed no later than their due-dates). In SUBSET SUM problem we are given a finite set of integer numbers $C = \{c_1, c_2, \ldots, c_n\}$ and an integer number $B \leq \sum_{i=1}^{n} c_i$.

This decision problem gives a “yes” answer iff there exists a subset of $C$ which sums up to $B$. Given an arbitrary instance of SUBSET SUM, we construct our scheduling instance with $n + 1$ jobs with the total length of $\sum_{i=1}^{n} c_i + 1$ as follows.

We have $n$ partition jobs $1, \ldots, n$ released at time $r_i = 0$ with $p_i = c_i$, $r_i = 0$ and $d_i = \sum_{i=1}^{n} c_i + 1$, for $i = 1, \ldots, n$. Besides these partition jobs, we have another separator job $I$, released at time $r_I = B$ with $p_I = 1$, $r_I = r_I$ and with the due-date $d_I = B + 1$. Note that this transformation creating an instance of $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{max}$ is polynomial as the number of jobs is bounded by the polynomial in $n$, and all magnitudes can be represented in binary encoding in $O(n)$ bits.

Now we prove that there exists a feasible schedule in which all jobs meet their due-dates iff there exists a solution to our SUBSET SUM problem. In one direction, suppose there is a solution to SUBSET SUM formed by the numbers in set $C \subseteq C$. Let $J'$ be the set of the corresponding partition jobs in our scheduling instance. Then we construct a feasible schedule in which all jobs meet their due-dates as follows (see Figure 5). We first schedule the partition jobs from set $J'$ in the interval $[0, B]$ in a non-delay fashion (using for instance, Jackson's heuristic). Then we schedule the separator job at time $B$ completing it by its due-date $B + 1$ and then the rest of the partition jobs starting from time $B + 1$ again in a non-delay fashion by Jackson’s heuristic. Observe then that the latest scheduled job will be completed exactly at time $\sum_{i=1}^{n} c_i + 1$ and all the jobs will meet their due-dates. Therefore, there exists a feasible schedule in which all jobs meet their due-dates whenever SUBSET SUM has a solution.
In the other direction, suppose there exists a feasible schedule $S$ in which all jobs meet their due-dates. Then in that schedule, the separator job must be scheduled in the interval $[B, B + 1]$, whereas the latest scheduled partition job must be completed by time $\sum_{i=1}^{n} c_i + 1$. But this may only be possible if the interval $[0, B]$ is completely filled in by the partition jobs, i.e. there must be a corresponding subset $J'$ of the partition jobs that fill in completely the interval $[0, B]$ in schedule $S$ (if this interval were not completely filled out in schedule $S$ then the completion time of the latest scheduled partition job would be $\sum_{i=1}^{n} c_i + 1$ plus the total length of the gap within the interval $[0, B]$ and hence that job would not meet its due-date (seeas illustrated in Figure 6). Now clearly, the processing times of the jobs in set $J'$ form the corresponding solution to SUBSET SUM. \[\blacksquare\]

5. Tractable special cases of problem $1|\{r_1, r_2\}, \{q_i\}|C_{\text{max}}$

In Section 4 we have shown that the version of our general problem with two job release times, i.e., the problem $1|\{r_1, r_2\}, \{q_i\}|C_{\text{max}}$ is NP-hard. In this section we establish some useful properties of an optimal solution to this problem that can be verified in time $O(n \log n)$.

Assume that the sequence of jobs $\{j_i\}$ is enumerated so that the first $m$ jobs of the sequence are ones released at time $r_1$ and the next $n - m$ jobs are ones released at time $r_2$ ($m < n$). We let $J_1$ and $J_2$ stand for the set of jobs released at the time $r_1$ and $r_2$, respectively.

**Lemma 7** If $\sum_{i=1}^{m} p_i + r_1 \leq r_2$ then $\sigma$ is optimal.

*Proof* Since $\sum_{i=1}^{m} p_i + r_1 \leq r_2$, $J$-heuristic in the first $m$ iterations will schedule all the jobs released at time $r_1$. As the jobs $j_i, j = 1, \ldots, m$) are released simultaneously at the time $r_2$ they are scheduled optimally by Lemma 3.

By the condition in the Lemma, all jobs released at the time $r_1$ are completed by time $r_2$, and hence all the jobs in set $J_2$ are available for scheduling at time $r_2$. Then again $J$-heuristic will schedule all these jobs optimally (Lemma 3).

Now clearly, by pasting together the two above partial schedules, we obtain a complete optimal schedule (as if there arises a gap between the two portions of that schedule then it is unavoidable). \[\blacksquare\]

**Lemma 8** Let $k < m$ be such that for the first $k$ jobs with the largest tails in schedule $\sigma$ the equality $\sum_{i=1}^{k} p_i = r_2 - r_1$ holds. Then $\sigma$ is optimal.

*Proof* We distinguish the following two cases based on the fact that for the first $k$ jobs with the largest tails released at time $r_2$, $\sum_{i=1}^{k} p_i = r_2 - r_1$ is satisfied:

**Case 1:** $k = m$. In this case $\sum_{i=1}^{m} p_i = r_2 - r_1$ and by Lemma 7, schedule $\sigma$ is optimal (with the equality being hold for any tow neighboring jobs).

**Case 2:** $k < m$. Let $\sigma'$ be the $J$-schedule for the first $k$ jobs. Since all these jobs are released at the time $r_2$, they are scheduled optimally by Lemma 3. Further, since $\sum_{i=1}^{k} p_i \leq r_2 - r_1$, the earliest starting time of the jobs $i_{k+1}, \ldots, i_m$ is $r_2$. Hence by time $r_2$ all the remaining yet unscheduled $n - k$ jobs become simultaneously available. Let $\sigma''$ be the $J$-schedule for these jobs. Since all yet unscheduled jobs are available by time $r_2$, $\sigma''$ is optimal by Lemma 3.

Now let $\sigma$ be the $J$-schedule obtained joining the $J$-schedules $\sigma'$ and $\sigma''$. Note that schedule $\sigma$ has no gap, hence it consists of a single block. By our construction, there exists no emerging job in schedule.
σ, and it is optimal by Lemma 2.

Below we let i, i ≤ k, be the earliest arisen job from J during the construction of schedule σ, such that τ + p > τ₂, and we let q′ be the largest tail among the tails of the jobs released at time τ₂.

**Lemma 9** If q′ ≥ q, then σ is optimal.

**Proof** As it is easily seen, because q′ ≥ q, schedule σ contains no emerging job and hence is optimal by Lemma 2.

Due to Theorem 1, from now on, let us assume that α(σ) ∈ J and that B(σ) contains at least one emerging job (otherwise, schedule σ is optimal by Lemma 2.

**Observation 2** K(σ) ⊂ J

**Proof** By the contradiction, suppose that kernel K(σ) contains at least one job j ∈ J. Then for all i ∈ J, scheduled before kernel K(σ), q′ ≥ q by J-heuristic. As α(σ) is the job with smallest tail of kernel K(σ) (by the J-heuristic and the definition of kernel K(σ)), q′ ≥ qo and hence, q′ ≥ qo. Then schedule σ may have no emerging job. We came to a contradiction proving our claim.

**Observation 3** For problem 1|(r, r₂), (q, q₂)Cmax E(σ) ∈ J and for any emerging job e ∈ J, q′ ≥ q.

**Proof** The first statement follows from the fact that the earliest scheduled job of kernel K(σ) in schedule σ belongs to set J which, in turn, follows from the J-heuristic. The second claim also follows from J-heuristic which schedules all jobs released at time τ (in particular, the jobs from E(σ) in non-increasing order of their tails (in particular, up to time τ, before the jobs of kernel K(σ)), whereas job l is the latest scheduled one from set e(σ) in schedule α.

**Observation 4** The kernel K(σ) contains jobs with the largest tails from set J scheduled in the non-increasing order of their tails.

**Proof** By Observation 3, E(σ) ⊂ J. By definition the live emerging job, l is the last job scheduled before kernel K(σ) released at time τ and the first job with completion time greater than or equal to τ. Then for all e ∈ J, scheduled before job l in schedule σ (job l starts strictly before time τ), there may exist no job of set J scheduled before job l in schedule σ (job l starts strictly before time τ), whereas job l is the latest scheduled one from set e(σ) in schedule α. Therefore, for all jobs e ∈ J, scheduled after kernel K(σ), it follows that the tail of the job released at time τ and the first job with completion time greater than or equal to τ. Then for all e ∈ J, scheduled before job l in schedule σ (job l starts strictly before time τ), there may exist no job of set J scheduled before job l in schedule σ (job l starts strictly before time τ), whereas job l is the latest scheduled one from set e(σ) in schedule α.

Recall that, for a given J-schedule S, the complementary schedule S always contains a gap between the jobs of kernel K(S), i.e. the earliest scheduled job of K(S) starts at its release time in schedule S. In general, for any emerging job e ∈ E(S), the complementary schedule S may contain a gap or not, depending on the length of that job (compared to that of job l). We will see this in more details later.

**Observation 5** The order of the jobs scheduled after time τ and before and after job e (particularly that of the jobs of K(σ)) is the same in both schedules σ and σ′.

**Proof** Note that since all jobs of kernel K(σ) and the jobs scheduled after this kernel in σ were released before time τ, J-heuristic should have been scheduled these jobs in the non-increasing order of their tails in schedule σ. Then J-heuristic will also schedule all the former jobs in the same order in schedule σ′, i.e. the jobs before job e and after that job (also released by time τ), in particular, ones of kernel K(σ′) will be included in the same order in both schedules.

Let now J[e] be the set of all jobs j scheduled after τ in σ such that q < q < qo, It follows that all jobs from set J[e] are scheduled immediately after kernel K(σ) and before job e in σ′.
Observation 6 \( J(e) \subseteq J_2 \)

Proof First note that set \( J[e] \) has no job with the tail equal to \( q_e \). Besides, \( K(\sigma) \subseteq J_2 \) and \( q_i > q_e \) for every job \( k \in K(\sigma) \). Furthermore, by \( J \)-heuristic, for every job \( j \in J_1 \) scheduled in \( \sigma \) before \( K(\sigma) \), \( q_j \geq q_e \), and for every job \( i \in J_2 \) scheduled after \( K(\sigma) \), \( q_i \leq q_e \), hence \( q_i \leq q_e \) as \( q_e \geq q_i \) and the Observation follows from the definition of set \( J[e] \).

Proposition 1 Without loss of generality, it might be assumed that all the jobs \( j \) with \( q_j = q_e \) scheduled after kernel \( K(\sigma) \) in \( \sigma \) are included behind job \( e \) in complementary schedule \( \sigma_e \).

Observation 7 In schedule \( \sigma_e \) the jobs of kernel \( K(\sigma) \) and those of set \( J[e] \) are left-shifted by the same amount, whereas all jobs \( j \) with \( q_j \leq q_e \) are right-shifted also by the same amount.

Proof Note that while constructing schedule \( \sigma_e \), \( J \)-heuristic schedules all the jobs in the non-increasing order of their tails behind time \( r_e \) (as all of them are released by that time moment). While constructing schedule \( \sigma_e \), the same set of jobs plus job \( e \) are available behind time \( r_e \). In particular, it is easy to see that if \( p_e \geq \Delta_e \), all jobs of kernel \( K(\sigma) \) and those from set \( J[e] \) will be left-shifted by \( \Delta_e \) and if \( p_e < \Delta_e \), this left-shift will equal to \( p_e \), whereas any \( j \) with \( q_j \leq q_e \) will be scheduled behind job \( e \) and will be right-shifted correspondingly.

Lemma 10

\( (1) \) If \( p_e > \Delta_e \), then \( \sigma_e \) has a gap of length \( p_e - \Delta_e \).

\( (2) \) If \( p_e \leq \Delta_e \), then \( \sigma_e \) has no gap.

Proof First note that \( \Delta_e \) is now \( c_e(\sigma) - r_e \). By the definition of \( \sigma_e \), no job \( j \in J_1 \) with \( q_j < q_e \) scheduled after \( K(\sigma) \) in \( \sigma \) will be scheduled before \( K(\sigma) \) in \( \sigma_e \). If \( p_e \geq \Delta_e \), in schedule \( \sigma_e \), the jobs of kernel \( K(\sigma) \) and of set \( J[e] \) will have the left-shift of length \( \Delta_e \) (which is the maximum possible by Observation 1). Then \( \sigma_e \) will have a gap of length \( p_e - \Delta_e \). If \( p_e < \Delta_e \), the jobs of kernel \( K(\sigma) \) and those of set \( J[e] \) will have a left-shift of length \( p_e \), hence \( \sigma_e \) will have no gap.

Due to Lemma 10 we define the gap \( \delta_e \) in schedule \( \sigma_e \) as follows:

\[
\delta_e = \begin{cases} 
0 & \text{if } p_e \leq \Delta_e \\
 p_e - \Delta_e & \text{if } p_e > \Delta_e
\end{cases}
\]

for all \( e \in E(\sigma) \).

Lemma 11 The full completion time of job \( e \) in \( \sigma_e \)

\[
C_e(\sigma_e) = C_{\sigma_e}(\sigma) + \sum_{j \in J[e]} p_j + \delta_e + q_e 
\]

Proof By the definition of the set \( J[e] \), job \( e \) will be scheduled in \( \sigma_e \) after all jobs of \( J[e] \) and before of all jobs \( i \) with \( q_i = q_e \) (see Proposition 1). By the Observation 1, \( \Delta_i \) is the maximum possible left-shift for the jobs of \( K(\sigma) \) in \( \sigma_e \). The full completion time of job \( e \) in \( \sigma_e \) will depend on the amount of this left-shift. Consider the following two cases. If \( p_e > \Delta_e \), then the jobs of \( K(\sigma) \) and those of set \( J[e] \) will be left-shifted by the amount \( \Delta_i \) in schedule \( \sigma_e \). Recall that the last job of kernel \( K(\sigma) \) scheduled in \( \sigma \) is \( \sigma(\sigma) \). Then,

\[
C_e(\sigma_e) = (C_{\sigma\sigma}(\sigma) + \sum_{j \in J[e]} p_j) - \Delta_i + p_e + q_e 
\]

Let \( \delta_e = p_e - \Delta_e \), then \( p_e = \delta_e + \Delta_e \). Therefore
Note that in this case \( \sigma_e \) has a gap of length \( \delta_e \).

If \( p_e \leq \Delta_i \), then by the definition of set \( J(e) \) and the construction of schedule \( \sigma_e \), the jobs of kernel \( K(\sigma) \) and those of set \( J(e) \) will be left-shifted by amount \( p_e \) in schedule \( \sigma_e \). Therefore,

\[
C_e(\sigma_e) = C_{o\sigma}(\sigma) + \sum_{j \in J(e)} p_j - \Delta_i + \delta_e + \Delta_l + q_e
\]

\[
= C_{o\sigma}(\sigma) + \sum_{j \in J(e)} p_j + q_e.
\]

In this case, \( \sigma_e \) has a gap of length \( \delta_e \).

Note that in this case \( \sigma_e \) has no gap.

Consider the following inequality:

\[
c_{o\sigma}(\sigma) + \sum_{j \in J(e)} p_j + \delta_e + q_e \geq C_{o\sigma}(\sigma),
\]

(1)

**Lemma 12** If \( p_e \geq \Delta_i \) and the inequality (1) for job \( e \in E(\sigma) \) holds, then in \( S_{opt} \), job \( e \) is scheduled before kernel \( K(\sigma) \).

**Proof** As \( p_e \geq \Delta_i \), \( \sigma_e \) has a gap by Lemma 10 and kernel \( K(\sigma) \) starts at its earliest starting time \( r(K(\sigma)) = r_2 \) in schedule \( \sigma_e \). By inequality (1), \( C_e(\sigma_e) \geq C_{o\sigma}(\sigma) \); hence, \( |\sigma_e| \geq |\sigma| \). Then it follows that job \( e \) cannot be scheduled after kernel \( K(\sigma) \) in schedule \( S_{opt} \) and the lemma is proved.

**Theorem 3** If \( p_e \geq \Delta_i \) for every \( e \in E(\sigma) \) and the inequality (1) is satisfied for job \( e \), then \( \sigma \) is optimal.

**Proof** Recall that if schedule \( \sigma \) is not optimal, kernel \( K(\sigma) \) must be restarted earlier. This will be possible only if at least one job \( e \in E(\sigma) \) is rescheduled after kernel \( K(\sigma) \). Suppose \( \sigma_e \) is a schedule with a better (smaller) makespan than schedule \( \sigma \). Then similarly, it is straightforward to verify that, by the condition of the Theorem, inequality (1) yields \( C_e(\sigma_e) \geq C_{o\sigma}(\sigma) \) for every \( e \in E(\sigma) \). Then by Lemma 12, schedule \( \sigma \) must be optimal.

Let \( E(S, l) = \{ e \in E(S) \setminus \{ l \} \mid p_e \geq p_l \} \).

**Lemma 13** If the inequality (1) is satisfied for job \( l \), then in \( S_{opt} \), any job \( e \in E(\sigma, l) \) is scheduled before kernel \( K(\sigma) \).

**Proof** By Lemma 12, \( l \) is scheduled in \( S_{opt} \) before kernel \( K(\sigma) \). Let \( k \) be the last job scheduled of \( J(l) \) in \( \sigma \) (recall that \( J(l) \) is the set of all jobs \( j \) scheduled after \( r_j \) in \( \sigma \) such that \( q_i < q_j < q_{opt} \)). By J-heuristic, job \( k \) has the tail less than the jobs of set \( J(l) \). By the definition of the set \( J(l) \), without loss of generality we assume that the J-heuristic schedules job \( l \) after job \( k \) in \( \sigma_e \). In addition, the starting time of job \( l \) must be the completion time of job \( k \) in \( \sigma_e \), i.e. \( t_k(\sigma_e) = c_k(\sigma_e) \) (by Proposition 1 and because by J-heuristic, for any job \( i \) scheduled after of \( k \) in \( \sigma \), \( q_i < q_k \) and therefore \( q_i < q_{opt} \) by the definition of set \( J(l) \)).

It is easy to see that for all \( e \in E(\sigma, l) \), a \( \delta_e \geq \delta_l \) (because \( p_e \geq p_l \) and by the definition of \( \delta_l \)). Besides, \( k \) is scheduled after jobs of kernel \( K(\sigma) \) in \( \sigma_e \) and job \( e \) can be scheduled before or after job \( k \) in \( \sigma_e \). We respectively distinguish the following two cases. If job \( e \) is scheduled in \( \sigma_e \) after of \( k \), the starting time of job \( e \) in schedule \( \sigma_e \) is equal to the starting time of job \( l \) in schedule \( \sigma_e \), i.e. \( t_e(\sigma_e) = t_l(\sigma_e) \) since for any job \( i \) scheduled after job \( l \) in \( \sigma_e \), \( a_i \leq q_l \) and \( q_i < q_{opt} \) since \( q_{opt} \geq q_l \). As the inequality (1) is satisfied for \( l \),
it is also satisfied for job e because $\delta_e \geq \delta_i$ and $q_e \geq q_r$. By Lemma 12, e is scheduled before $K(\sigma)$ in $S_{opt}$.

If job e is scheduled in $\sigma_2$ before job k, the jobs scheduled after job e in schedule $\sigma_e$ (in particular the jobs of set $J[|e|]$) are right-shifted, therefore, the completion time of job k in $\sigma_e$ is greater than the completion time of job k in $\sigma$. Also, because $\delta_e \geq \delta_i$, the completion time of job k in $\sigma_e$ is greater than or equal to the completion time of job l in $\sigma$, ($c_i(\sigma_e) \geq c_i(\sigma)$). As $q_i > q_e$, $c_i(\sigma_e) > c_i(\sigma)$ and therefore $|\sigma_e| > |\sigma|$. Then job e cannot be scheduled after kernel $K(\sigma)$ in schedule $S_{opt}$ since job l satisfies the inequality (1), hence $|\sigma_e| > |\sigma|$.

We have shown that job e cannot be scheduled after kernel $K(\sigma)$ in schedule $S_{opt}$ and Lemma is proved.

From here on, we denote by k be the job with the greatest full time completion in schedule $\sigma$ among all the jobs scheduled after the jobs of set $J[|e|]$. As it can be easily seen, job k can be either from set $J_1$ or $J_2$.

**Observation 8** If the overflow job of schedule $\sigma_e$ is scheduled after $r_2$, then it belongs to set $\{o(\sigma), e, k\}$.

**Proof** From Observation 5 the jobs scheduled behind time $r_2$ and before job e in schedule $\sigma_e$ have the same processing order as in schedule $\sigma$. At the same time, by Observation 7 the jobs of kernel $K(\sigma)$ and those from set $J[e]$ have the same left-shift $p_e$ in schedule $\sigma_e$. Right after the latter jobs job e is included in schedule $\sigma_e$ (Proposition 1) and hence both, the processing order and the starting/completion times of the following jobs will remain unchanged. Now our claim follows from the definition of jobs $o(\sigma)$, e and k. $\square$

**Lemma 14** Schedule $\sigma_e$ is optimal if $p_e \geq \Delta_2$ and $o(\sigma) = o(\sigma_e)$.

**Proof** Since $p_e > \Delta_2$, the jobs of kernel $K(\sigma)$ are left-shifted in schedule $\sigma_e$ by the amount $\Delta_2$. By definition of $\Delta_2$ and from the fact that all jobs of kernel $K(\sigma)$ are released by time $r_2$, the earliest job from that kernel starts at its early starting time in $\sigma_e$, i.e. $t_K(\sigma_e) = r_2$. By the J-heuristic, for any job $i \in K(\sigma) \setminus \{o(\sigma)\}, q_i \geq q_{\text{opt}}$. It follows that the completion time of job $o(\sigma)$ in schedule $\sigma_e$ is the least possible and schedule $\sigma_e$ optimal. $\square$

**Theorem 4** If job $o(\sigma_e)$ is scheduled before kernel $K(\sigma)$, then schedule $\sigma_e$ is optimal.

**Proof** Since $K(\sigma) \subset J_2$ (Observation 2) and first job of $K(\sigma)$ is the job with the largest tail in set $J_2$, all jobs scheduled before kernel $K(\sigma)$ are in set $J_2$. Hence the overflow job $o(\sigma_e)$ is in set $J_e$. By the J-heuristic, all jobs scheduled before kernel $K(\sigma)$ are scheduled in the non-increasing order of their tails (see also Lemma 3). Then there may exist no schedule $\sigma_\epsilon$ such that $|\sigma_\epsilon| < |\sigma_e|$ and $\sigma_\epsilon$ is optimal. $\square$

Below we illustrate the above Theorem using a problem instance of our problem with 5 jobs. Schedules $\sigma$ and $\sigma_e$ are illustrated in Figures 7 and 8, respectively (the numbers within the circles stand for the full completion times of the corresponding jobs).

**Observation 9** If in schedule $\sigma_e$ the overflow job $o(\sigma_e)$ is scheduled before kernel $K(\sigma)$, then $o(\sigma_e) \notin E(\sigma) \setminus \{e\}$.

**Proof** Indeed, by the definition of an emerging job, all jobs from set $E(\sigma)$ have the tail smaller than that of the jobs in kernel $K(\sigma)$, whereas these jobs, except job e, are scheduled before the jobs of that kernel in schedule $\sigma_e$. Then clearly, none of them (excluding job e) may be the overflow job in schedule $\sigma_e$. $\square$
Figure 7. Schedule $\sigma$.

\[
\begin{align*}
  r_1 &= 0 & p_1 &= 5 & q_1 &= 40 \\
  r_2 &= 0 & p_2 &= 4 & q_2 &= 30 \\
  r_3 &= 0 & p_3 &= 10 & q_3 &= 8 \\
  r_4 &= 10 & p_4 &= 10 & q_4 &= 16 \\
  r_5 &= 10 & p_5 &= 6 & q_5 &= 11
\end{align*}
\]

Figure 8. Schedule $\sigma_1$, in this schedule the overflow job $o(\sigma_1)$ is scheduled before kernel $K(\sigma)$. 

\[
\begin{align*}
  o(\sigma_1) &= 45 \\
  o(\sigma) &= 46
\end{align*}
\]
5.1 A few more properties for equal-length jobs in set $J_i$

In this section we consider another special case of our problem when all jobs released at time $r_i$ have the same processing time $p$. We abbreviate this problem as $1\{r_1, r_2\}, q_i, p_{r_i,j} = p$ $C_{max}$ ($p_{r_i,j} = p$ denotes that all jobs released at the time $r_i$ have processing time $p$).

Recall that the left-hand side in inequality (1) is the full completion time of job $e$ in schedule $\sigma_e$ (job $k$ is defined as earlier). We define set $J(e,k) = \{i|q_e \leq q_i \leq q_e\}$.

**Lemma 15** The full completion time of job $k$ in the schedule $\sigma_e$ is:

$$C_k(\sigma_e) = c_{e}(\sigma) + \sum_{j \in J(e,k)} p_j + \delta_e + \sum_{j \in J(e,k)} p_i + p_k + q_e.$$

*Proof* Is analogous to that of Lemma 11. $\square$

Consider the following inequality:

$$C_{\sigma(e)}(\sigma) + \sum_{j \in J(e)} p_j + \delta_e + \sum_{j \in J(e,k)} p_i + p_k + q_e \geq C_{\sigma(e)}(\sigma)$$

(2)

**Lemma 16** For problem $1\{r_1, r_2\}, q_i, p_{r_i,j} = p$ $C_{max}$ in any schedule $\sigma_e$ there occurs a gap before kernel $K(\sigma)$. Moreover, $\delta_e = \delta_i$ for every job $e \in E(\sigma) \setminus \{l\}$.

*Proof* We have $p_e > \Delta_i$, since $p_e = p_i = p$ for all $e \in E(\sigma) \setminus \{l\}$. By Lemma 10, schedule $\sigma_e$ has a gap of length $p_e - \Delta_i$ and $\delta_e = \delta_i$ for all $e \in E(\sigma) \setminus \{l\}$. $\square$

**Lemma 17** If the full completion time of job $k$ satisfies the inequality (2), then each job $e \in E(\sigma)$ is scheduled before kernel $K(\sigma)$ in $S_{opt}$.

*Proof* The inequality (2) implies that $C_k(\sigma) \geq C_{\sigma(e)}(\sigma)$; hence, $|\sigma| \geq |\sigma|$. As the full completion time of $k \in \sigma$ is less than the full completion time of the overflow job $\sigma(\sigma)$ (otherwise, job $k$ would be the overflow job in $\sigma$) and $C_k(\sigma) \geq C_{\sigma,e}(\sigma)$, job $l$ cannot be scheduled after kernel $K(\sigma)$ in schedule $S_{opt}$. By Lemma 16, $\delta_e = \delta_i$ for every emerging job $e$. Then the starting time of job $k$ in schedule $\sigma_e$ is the same as that in $\sigma$. Therefore $C_k(\sigma) \geq C_{\sigma,e}(\sigma)$ and $|\sigma| \geq |\sigma|$ for each job $e \in E(\sigma)$. It follows that no job $e \in E(\sigma)$ can be scheduled after kernel $K(\sigma)$ in schedule $S_{opt}$. $\square$

**Theorem 5** For problem $1\{r_1, r_2\}, q_i, p_{r_i,j} = p$ $C_{max}$ schedule $\sigma$ is optimal, if:

1. The inequality (1) is satisfied for job $l$ in schedule $\sigma_l$
2. The inequality (2) is satisfied for job $k$ in schedule $\sigma_k$

*Proof* For the first claim, by Lemma 13, any job $e \in E(\sigma)$ is scheduled in $S_{opt}$ after kernel $K(\sigma)$. This implies that every job $e \in E(\sigma) \setminus \{l\}$ satisfies the inequality (1) and by Theorem 3, $\sigma$ is optimal.

As to the second claim, by Lemma 17 every job $e \in E(\sigma) \setminus \{l\}$ must be scheduled before kernel $K(\sigma)$ in $S_{opt}$ hence $\sigma$ is optimal. $\square$

5.2 Short emerging jobs

In this subsection we study the case when the use of a short emerging job, i.e., one with processing time less than $\Delta_i$, leads us to useful optimality properties. Recall from Lemma 10 that for such a short job $e$ schedule $\sigma_e$ has no gap. Throughout this subsection we let $j$ be the job from set $J[e]$ scheduled...
the last in schedule \( \sigma \), and, as earlier, \( k \) stands for the job with the maximum full completion time among the jobs scheduled behind the jobs of \( J[e] \) in schedule \( \sigma \).

**PROPERTY 1** Let \( p_e \leq \Delta_e \). If \( J[e] \neq \emptyset \) then \( c_e(\sigma_e) = c_j(\sigma), \) otherwise \( (J[e] = \emptyset) c_e(\sigma_e) = c_{o(e)}(\sigma) \).

**Proof** Suppose first \( J[e] \neq \emptyset \). Consider the set of the emerging jobs from schedule \( \sigma \) included in between job \( e \) and kernel \( K(\sigma) \), the jobs of kernel \( K(\sigma) \) and ones from set \( J[e] \). Note that these are the jobs which will be included before job \( e \) in schedule \( \sigma_e \). These jobs will be left-shifted by \( p_e \) time units in the latter schedule (see Lemma 10). Since job \( j \) is one of these jobs, it will also be left-shifted by the same amount and job \( e \) will start immediately after after job \( j \) in schedule \( \sigma_e \). If there are jobs scheduled after kernel \( \sigma \) with the tail equal to \( q_e \), they will be included behind job \( e \) by Proposition 1. It follows that the completion time of job \( e \) in schedule \( \sigma_e \) is equal to that of job \( j \) in schedule \( \sigma \). This completes the case \( J[e] \neq \emptyset \), and the other case \( J[e] = \emptyset \) can be similarly seen.

**LEMMA 18** If \( p_e < \Delta_e \) then \( c_e(\sigma_e) < c_{o(e)}(\sigma) \).

**Proof** By Property 1, \( c_e(\sigma_e) = c_j(\sigma) \) if \( J[e] \neq \emptyset \) and \( q_e < q_j \) (by definition of set \( J[e] \)). Hence, \( c_e(\sigma_e) < c_j(\sigma) \). But \( c_j(\sigma) < c_{o(e)}(\sigma) \) and our claim follows. If now \( J[e] = \emptyset \), by Property 1, \( c_e(\sigma_e) = c_{o(e)}(\sigma) \) and \( q_e < q_{o(e)} \) (by definition of overflow job). Hence, \( c_e(\sigma_e) < c_{o(e)}(\sigma) \).

**OBSERVATION 10** If \( p_e < \Delta_e \) then \( c_e(\sigma_e) = c_j(\sigma) \).

**Proof** Since schedule \( \sigma_e \) has no gap (Lemma 10) all the emerging jobs scheduled between \( e \) and kernel \( K(\sigma) \), the jobs of kernel \( K(\sigma) \) and ones from set \( J[e] \) will have the same left-shift of length \( p_e \) in schedule \( \sigma_e \). At the same time, all jobs scheduled after jobs of set \( J[e] \) in schedule \( \sigma \) will have no shift in schedule \( \sigma_e \) as job \( e \) will be included before them. Then the starting time of job \( k \) in both schedules \( \sigma_e \) and \( \sigma \) is the same and our claim follows.

**THEOREM 6** If \( p_e < \Delta_e \) then \( |\sigma_e| \leq |\sigma| \).

**Proof** In schedule \( \sigma_e \), either the overflow job \( o(\sigma_e) \) is scheduled before or behind kernel \( K(\sigma) \). For the latter case, by Observation 8, the only potential overflow jobs in schedule \( \sigma_e \) are jobs \( o(\sigma), e \) and \( k \). Since the jobs of kernel \( K(\sigma) \) have the left-shift of length \( p_e \) in schedule \( \sigma_e \), \( C_{o(e)}(\sigma_e) = C_{o(e)}(\sigma) - p_e \) and hence \( C_{o(e)}(\sigma_e) < C_{o(e)}(\sigma) \). By Lemma 18, \( C_e(\sigma_e) < C_{o(e)}(\sigma) \), and by Observation 10, \( C_e(\sigma_e) = C_j(\sigma) \). But \( C_j(\sigma) \leq C_{o(e)}(\sigma) \) and hence \( C_e(\sigma_e) \leq C_{o(e)}(\sigma) \). Thus the full completion time of jobs \( o(\sigma_e), e \) and \( k \) in schedule \( \sigma_e \) is no more than that in schedule \( \sigma \) and the theorem holds for first second case above.

For the first case above (the overflow job \( o(\sigma_e) \) is scheduled before kernel \( K(\sigma) \)), again \( |\sigma_e| \leq |\sigma| \). Indeed, any job scheduled before kernel \( K(\sigma) \) in schedule \( \sigma \) must have the full completion time no more than job \( o(\sigma) \) (by the definition of the overflow job) whereas the set of all jobs scheduled before kernel \( K(\sigma) \) in schedule \( \sigma \) is the same as that in schedule \( \sigma_e \) except that job \( e \) is rescheduled behind kernel \( K(\sigma) \) in schedule \( \sigma_e \).

**OBSERVATION 11** If \( p_e \leq \Delta_e \) and \( J[e] \neq \emptyset \), then the overflow job in schedule \( \sigma_e \) is:

1. Job \( k \) if \( C_k(\sigma) > C_{o(e)}(\sigma) - p_e \) and \( C_k(\sigma) > C_j(\sigma) + q_e \).
2. Job \( e \) if \( C_e(\sigma) + q_e > C_{o(e)}(\sigma) - p_e \) and \( C_e(\sigma) + q_e > C_k(\sigma) \).
3. Job \( o(\sigma) \) if \( C_{o(e)}(\sigma) - p_e > C_k(\sigma) \) and \( C_{o(e)}(\sigma) - p_e > C_j(\sigma) + q_e \).

**Proof** Below we prove each of these cases using the fact that the overflow job in schedule \( \sigma_e \) is in set \( \{o(\sigma), e, k\} \) (Observation 8).
(1) By the definition of the overflow job in $\sigma$, $C_{\text{overflow}}(\sigma) \geq C_k(\sigma)$. Since $p_e \leq \Delta_k$, job $\sigma(e)$ will have the left-shift of length $p_e$ in $\sigma$, i.e., $C_{\text{overflow}}(\sigma(e)) = C_{\text{overflow}}(\sigma) - p_e$. (By Observation 7 and Lemma 10). By Observation 10, $C_k(\sigma(e)) = C_k(\sigma)$, and the inequality $C_k(\sigma) > C_{\text{overflow}}(\sigma) - p_e$ implies that $C_k(\sigma(e)) > C_{\text{overflow}}(\sigma)$. By Property 1, $C_k(\sigma(e)) = C_k(\sigma)$. Hence, the full completion time of job $e$ in $\sigma$.

(2) From $C_{\text{overflow}}(\sigma(e)) = C_{\text{overflow}}(\sigma) - p_e$, and $C_e(\sigma(e)) = C_k(\sigma) + q_e$ (see the proof of the above claim 1), and from $C_k(\sigma) + q_e > C_{\text{overflow}}(\sigma) - p_e$, we get that $C_{\text{overflow}}(\sigma) > C_{\text{overflow}}(\sigma(e))$. From $C_{\text{overflow}}(\sigma(e)) = C_k(\sigma)$ (Observation 10) and $C_k(\sigma) + q_e > C_k(\sigma)$, $C_k(\sigma) > C_{\text{overflow}}(\sigma)$. Then $C_k(\sigma) > C_{\text{overflow}}(\sigma(e))$ and $C_k(\sigma(e)) > C_{\text{overflow}}(\sigma(e))$ yields that $e$ is the overflow job in $\sigma$.

(3) By the definition of the overflow job in $\sigma$, $C_{\text{overflow}}(\sigma) \geq C_k(\sigma)$, and we have seen that $C_{\text{overflow}}(\sigma(e)) = C_{\text{overflow}}(\sigma) - p_e$. By Observation 10, $C_k(\sigma(e)) = C_k(\sigma)$. Then from the inequality $C_{\text{overflow}}(\sigma) - p_e > C_k(\sigma)$, $C_{\text{overflow}}(\sigma(e)) > C_k(\sigma(e))$. We also know that $C_{\text{overflow}}(\sigma(e)) = C_k(\sigma) + q_e$. Then inequality $C_{\text{overflow}}(\sigma) - p_e > C_k(\sigma)$ implies that $C_{\text{overflow}}(\sigma) > C_k(\sigma)$. Thus similarly to the above two cases, $C_{\text{overflow}}(\sigma(e)) > C_k(\sigma(e))$ and $C_{\text{overflow}}(\sigma(e)) > C_k(\sigma(e))$ yields that job $\sigma(e)$ is the overflow job in $\sigma$.

Observation 12 If $p_e \leq \Delta_k$ and set $J[e] = \emptyset$, then the overflow job in schedule $\sigma$ is:

1. Job $k$ if $C_k(\sigma) > C_{\text{overflow}}(\sigma) - p_e$ and $C_k(\sigma) > C_{\text{overflow}}(\sigma) + q_e$.
2. Job $e$ if $C_{\text{overflow}}(\sigma) + q_e > C_{\text{overflow}}(\sigma) - p_e$ and $C_{\text{overflow}}(\sigma) + q_e > C_k(\sigma)$.
3. Job $\sigma(e)$ if $C_{\text{overflow}}(\sigma) - p_e > C_k(\sigma)$ and $C_{\text{overflow}}(\sigma) - p_e > C_{\text{overflow}}(\sigma) + q_e$.

Proof: Is analogous to that of Lemma 11.

Theorem 7 Schedule $\sigma$ is optimal if $p_e < \Delta_k$, $k \in J_1$ and $k = \sigma(\sigma)$.

Proof: Schedule $\sigma$ is better than schedule $\sigma$ due to Theorem 6. Moreover, we show that there may exist no better schedule than $\sigma$. It suffices to consider potential rearrangements in which only the emerging jobs are involved (see Lemma 2). To this end, note first that $C_{\text{overflow}}(\sigma) \geq C_k(\sigma)$. In other words, the former kernel $K(\sigma)$ need not be further left-shifted, whereas we show that job $k$ cannot be further left-shifted. Indeed, let $S$ be a schedule in which some emerging job(s) are rescheduled behind kernel $K(\sigma)$. Since $k \in J_1$, $q_e \geq q_e$ for any emerging job $e \in E(\sigma)$ (due to $H$-heuristic). Hence the $H$-heuristic will include any such emerging job before job $k$ in schedule $S$. Let us consider the following two possibilities.

1. If schedule $S$ has no gap, then because of the above observations and due to the fact that the set of jobs scheduled before job $k$ in both schedules $\sigma$ and $S$ is the same, job $k$ is started and completed in schedule $S$ at the same time as in schedule $\sigma$. Hence, $|S| = |\sigma|$.
2. If schedule $S$ has a gap, then since in schedule $S$ all the rescheduled emerging jobs are included before job $k$, the latter job will be right-shifted by the length of that gap. Then $C_k(S) > C_k(\sigma)$ and hence $|S| = |\sigma|$. We have showed that there may exist no schedule better than $\sigma$, and the proof is complete (Figure 9).

As we illustrate below with an instance of our problem, the above theorem cannot be extended to the case when $k \neq \sigma(\sigma)$ and $p_e \geq \Delta_k$, as then we are forced to solve the subset sum problem (see Section 4). Thus the problem bottleneck is reached when these conditions hold. The table below specifies our problem instance. In Figures 10 and 11, illustrating schedule $\sigma$ and an optimal schedule, respectively, for that instance, the numbers within the circles are the full completion times of the corresponding jobs.
Figure 9. Schedule $\sigma$, $|\sigma| = 57$, with the live emerging job $l = 4$ and $o(\sigma) = 9$, $J[l] = 5, 6$ and $k = 7.$

\[
\begin{array}{cccc}
 l & p_l & q_l & r_l \\
 1 & 0 & 5 & 30 \\
 2 & 0 & 4 & 25 \\
 3 & 0 & 6 & 20 \\
 4 & 0 & 10 & 13 \\
 5 & 18 & 7 & 16 \\
 6 & 0 & 3 & 10 \\
 7 & 0 & 8 & 6 \\
 8 & 18 & 4 & 27 \\
 9 & 18 & 3 & 25 \\
 10 & 18 & 2 & 3 \\
\end{array}
\]  
\[
\begin{array}{cccc}
 l & p_l & q_l & r_l \\
 1 & 0 & 5 & 30 \\
 2 & 0 & 4 & 25 \\
 3 & 0 & 6 & 20 \\
 4 & 0 & 10 & 13 \\
 5 & 18 & 7 & 16 \\
 6 & 0 & 3 & 10 \\
 7 & 0 & 8 & 6 \\
 8 & 18 & 4 & 27 \\
 9 & 18 & 3 & 25 \\
 10 & 18 & 2 & 3 \\
\end{array}
\]

Figure 10. Schedule $\sigma_i$, has the makespan 59 and it is not optimal as $|\sigma_i| > |\sigma|$. Here inequality (1) is not satisfied for job $l$, job $k = 7$ is the overflow job in $\sigma_i$. 

\[
\begin{array}{cccc}
 c_1 & p_1 & q_1 & r_1 \\
 0 & 5 & 30 & 0 \\
 2 & 4 & 25 & 2 \\
 3 & 6 & 20 & 3 \\
 4 & 10 & 13 & 4 \\
 5 & 7 & 16 & 5 \\
 6 & 3 & 10 & 6 \\
 7 & 8 & 6 & 7 \\
 8 & 4 & 27 & 8 \\
 9 & 3 & 25 & 9 \\
 10 & 2 & 3 & 10 \\
\end{array}
\]  
\[
\begin{array}{cccc}
 c_1 & p_1 & q_1 & r_1 \\
 0 & 5 & 30 & 0 \\
 2 & 4 & 25 & 2 \\
 3 & 6 & 20 & 3 \\
 4 & 10 & 13 & 4 \\
 5 & 7 & 16 & 5 \\
 6 & 3 & 10 & 6 \\
 7 & 8 & 6 & 7 \\
 8 & 4 & 27 & 8 \\
 9 & 3 & 25 & 9 \\
 10 & 2 & 3 & 10 \\
\end{array}
\]
6. Conclusion

Based on the analysis of the J-schedules, we have established formal conditions when an optimal solution can be efficiently obtained. Our analysis yielded strict explicit relationships and polynomial time solution of these particular cases. We have conducted a similar study for an NP-hard spatial case of our generic problem with only two allowable job release times. Due to our conjecture of adaptive production planning, these results may stimulate the development of more efficient production planning allowing a beneficial interaction between scheduling and pre-scheduling phases for the scheduling models with release and delivery times, in particular.

Our study essentially relied on the partition of the whole scheduling horizon into two basic types of intervals containing urgent (kernel) and non-urgent (emerging) jobs, and extracting and treating the conflicts occurring between these two types of intervals. Such an analysis give some practical intuition on the general perspectives and potential conflicts that may occur while using simple heuristic rules for the solution of the problem at scheduling phase. Based on the derived dominance rules, one may assert when such conflicts may be efficiently resolved, and how. A practitioner may either choose to follow these rules for particular real-life instances or even modify some data that might be altered to avoid these conflicts and obtain an optimal solution for a modified instance, adjusting the production to the conditions and dominance rules. A practitioner may also choose to go deeper and expand the given optimality conditions and dominance relations into more complex heuristic methods depending on the nature of the problem instances that arise in his/her particular application.

For future work, it would be interesting to extend the presented results (limited to a single-machine environment) to more general scheduling models such as multiprocessor and shop scheduling.

Funding
This work was supported by PRODEP for this research.

Author details
Elisa Chinos1
E-mail: meco@uaem.mx
Nodari Vakhania1
E-mail: nodari@uaem.mx
1 Centro de Investigación en Ciencias, UAEmor, Cuernavaca, Mexico.

† Part of this work, in a preliminary form, was presented at the 4th Int. Conference on Mathematical, Computational and Statistical Sciences (MCSS 2016) and Int. Conference on Abstract and Applied Analysis (ABAPAN 2016).

Citation information
Cite this article as: Adjusting scheduling model with release and due dates in production planning, Elisa Chinos & Nodari Vakhania, Cogent Engineering (2017), 4: 1321175.
Note
1. In this section \( n \) stands exclusively for the number of elements in SUBSET SUM; in the remaining part, \( n \) denotes the number of jobs.

Cover image
Source: Nodari Vakhania.

Corrigendum
This article was originally published with errors. This version has been corrected. Please see Corrigendum (https://doi.org/10.1080/23311916.2017.1336199).

References
Bratley, P., Florian, M., & Robillard, P. (1973). On sequencing with earliest start times and due-dates with application to computing bounds for \( \text{InlMIGF} \) problem. Naval Research Logistics Quarterly, 20, 57–67.

Brinkkoller, W., & Brucker, P. (2001). Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments. Journal of Scheduling, 4, 53–64.

Carlier, J. (1982). The one-machine sequencing problem. European Journal of Operational Research, 11, 42–47.

Carlier, J., & Pinson, E. (1989). An algorithm for solving job shop problem. Management Science, 35, 164–176.

Carlier, J., & Pinson, E. (1998). Jakson's pseudo preemptive schedule for the \( \text{Pm}\text{l},q,\text{C}^\text{max} \) problem. Annals of Operations Research, 83, 41–58.

Condotta, A., Knust, S., & Shakhlevich, N. V. (2010). Parallel batch scheduling of equal-length jobs with release and due dates. Journal of Scheduling, 13, 463–477.

Della Croce, F., & T'kindt, V. (2010). Improving the preemptive bound for the single machine dynamic maximum lateness problem. Operations Research Letters, 38, 589–591.

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. San Francisco, CA: Freeman.

Garey, M. R., Johnson, D. S., Simons, B. B., & Tarjan, R. E. (1981). Scheduling unit-time tasks with arbitrary release times and deadlines. SIAM Journal on Computing, 10, 256–269.

Gharbi, A., & Labidi, M. (2010). Jackson's semi-preemptive scheduling on a single machine. Computers & Operations Research, 37, 2082–2088.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 5, 287–326.

Hall, L. A., & Shmoys, D. B. (1992). Jackson's rule for single-machine scheduling: Making a good heuristic better. Mathematics of Operations Research, 17, 22–35.

Hoogeveen, J. A. (1995). Minimizing maximum promptness and maximum lateness on a single machine. Annals of Discrete Mathematics, 21, 100–114.

Jackson, J. R. (1955). Scheduling a production line to minimize the maximum tardiness. Management Science Research Project. Los Angeles: University of California.

McMahon, G., & Florian, M. (1975). On scheduling with ready times and due dates to minimize maximum lateness. Operations Research, 23, 475–482.

Nowicki, E., & Smutnicki, C. (1994). An approximation algorithm for a single-machine scheduling problem with release times and delivery times. Discrete Applied Mathematics, 48, 69–79.

Perregaard, M., & Clausen, J. (1998). Parallel branch-and-bound methods for the job-shop scheduling problem. Annals of Operations Research, 83, 137–160.

Potts, C. N. (1980). Analysis of a heuristic for one machine sequencing with release dates and delivery times. Operations Research, 28, 1436–1441.

Schrage, L. (1971, March). Obtaining optimal solutions to resource constrained network scheduling problems. Unpublished manuscript.

Simons, B. (1983). Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. SIAM Journal on Computing, 12, 294–299.

Simons, B., & Warmuth, M. (1989). A fast algorithm for multiprocessor scheduling of unit-length jobs. SIAM Journal on Computing, 18, 690–710.

Vakhania, N. (2003). A better algorithm for sequencing with release and delivery times on identical processors. Journal of Algorithms, 48, 273–293.

Vakhania, N. (2004). Single-machine scheduling with release times and tails. Annals of Operations Research, 129, 253–271.

Vakhania, N. (2009). Scheduling jobs with release times preemptively on a single machine to minimize the number of late jobs. Operations Research Letters, 37, 405–410.

Vakhania, N. (2012). Branch less, cut more and minimize the number of late equal-length jobs on identical machines. Theoretical Computer Science, 465, 49–60.

Vakhania, N. (2013). A study of single-machine scheduling problem to maximize throughput. Journal of Scheduling, 16, 395–403.

Vakhania, N., & Schepin, E. (2002). Concurrent operations can be parallelized in scheduling multiprocessor job shop. Journal of Scheduling, 5, 227–245.

Vakhania, N., Perez, D., & Corballo, L. Theoretical expectation versus practical performance of Jackson's heuristic. Mathematical Problems in Engineering, 2015, 10 pages. doi:10.1155/2015/484671