Polynomially solvable and NP-hard special cases for scheduling with heads and tails

Elisa Chinos, Nodari Vakhania
Centro de Investigaci´on en Ciencias, UAEMor, Mexico

Abstract— We consider a basic single-machine scheduling problem when jobs have release and delivery times and the objective is to minimize maximum job lateness. This problem is known to be strongly NP-hard. We study inherent structure of this problem exploring its special cases and the conditions when the problem can be efficiently solved.

Keywords– scheduling single-machine; heuristic; algorithm; release time; delivery time; due-date

I. INTRODUCTION

The *scheduling problems* deal with a finite set of requests called *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 machines so as to meet a given objective criteria.

In a basic single-machine scheduling problem that we consider in this paper we have one machine that can process n jobs. The *processing time* p_j of job j is the amount of time that it requires on the machine; job j is available from its *release time* r_j. The machine can handle at most one job at a time. Once job j completes on the machine, it still needs a (constant) *delivery time* q_j for its *full completion*. The delivery is machine-independent, i.e., it consumes no machine time (the jobs are delivered by an independent unit and this takes no machine time). The objective is to find a job sequence on the machine that minimizes the maximum job full completion time.

According to the conventional three-field notation introduced by Graham et al. [11] the above problem is abbreviated as $1|r_j, q_j|C_{\max}$: in the first field the single-machine environment is indicated, the second field specifies job parameters, and in the third field the objective criteria is given.

The problem has an equivalent formulation $1|r_j|L_{\max}$ in which delivery times are interchanged by *due-dates*. The due-date d_j of job j is the desirable time for the completion of job j. Whenever that job is completed on the machine behind its due-date, it is said to be *late*. In this setting the maximum job *lateness* $L_{\max}$, that is, the difference between the job completion time and its due-date, is to be minimized.

Given an instance of $1|r_j, q_j|C_{\max}$, one can obtain an equivalent instance of $1|r_j|L_{\max}$ 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 $1|r_j|L_{\max}$, an equivalent instance of $1|r_j, q_j|C_{\max}$ 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 $1|r_j, q_j|C_{\max}$ is minimized, the maximum job lateness in $1|r_j|L_{\max}$ is minimized, and vice-versa (see Bratley et al. [1] for more details).

Because of the above equivalence, we can use both above formulations interchangeably.

Our problem is known to be strongly NP-hard (Garey and Johnson [8]). For exact implicit enumerative algorithms see for instance, McMahon & Florian [15] and Carlier [3]. An efficient heuristic method that has been commonly used for problem $1|r_j, q_j|C_{\max}$ is the so-called Earliest Due-Date (EDD) heuristic proposed long time ago by Jackson [14] (see also Schrage [18]). Jackson's heuristic 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). For the sake of conciseness Jackson's heuristic has been commonly abbreviated as EDD-heuristic (Earliest Due-Date) or alternatively, LDT-heuristic (Largest Delivery Time). 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 $O(n \log n)$.

Jackson's 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. Potts [17] has proposed a modification of Jackson's heuristic for the problem $1|r_j, q_j|C_{\max}$. His algorithm repeatedly applies the heuristic $O(n)$ times and obtains an improved approximation ratio of $3/2$. Hall and Shmoys [12] have elaborated polynomial approximation schemes for the problem, and also an 4/3-approximation an algorithm for its version with the precedence relations with the same time complexity of $O(n^2 \log n)$ as the above algorithm from [17].

In a recent work [28], the maximum job processing time $p_{\max}$ is expressed as a fraction κ of the optimal objective value and a more accurate approximation ratio in terms of that fraction is derived. It was shown that Jackson's heuristic delivers a solution within a factor of $1 + 1/\kappa$ of the optimum. Such an estimation is useful, in practice, since it may drastically outperform the worst-case ratio of 2, as it was suggested by the computational experiments reported in the paper.

As to the special cases of our problem, 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 [13]. Garey et al. [9] 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 [22] was proposed an $O(n^2 \log n)$ algorithm for the minimization version with two pos-

sible job processing times.

For other related criteria, in [23] 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 [25] and [24], respectively, with time complexities $O(n^2 \log n)$ and $O(n^3 \log n \log p_{\max})$, respectively.

Our problem $1|r_j, q_j|C_{\max}$ has been shown to be useful also 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 $O(n^2 m)$ were proposed in Simons [19] and Simons & Warmuth [20], respectively. Using the same heuristic as a schedule generator in [21] was proposed an $O(q_{\max} mn \log n + O(m\nu n))$ algorithm for the minimization version of the latter problem, where $q_{\max}$ is the maximal job delivery time and $\nu < n$ is a parameter.

The problem $1|r_j, q_j|C_{\max}$ is commonly used for the solution of more complex job-shop scheduling problems. A solution of the former problem gives a string lower bound for job-shop scheduling problems. In the classical job-shop scheduling problem the preemptive version of Jackson's heuristic applied for a specially derived single-machine problem immediately gives a lower bound, see, for example, Carlier [3], Carlier and Pinson [4] and Brinkkotter and Brucker [2] and more recent works of Gharbi and Labidi [10] and Della Croce and T'kindt [7]. Carlier and Pinson [5] have used the extended Jackson's heuristic for the solution of the multiprocessor job-shop problem with identical machines, and it can also be adopted for the case when parallel machines are unrelated (see [26]). Jackson's heuristic can be useful for parallelizing the computations in scheduling job-shop Perregaard and Clausen [16], and also for the parallel batch scheduling problems with release times Condotta et al. [6].

The paper is organized as follows. xxxx In the next subsection we give a general overview of combinatorial optimization and scheduling problems mentioning the importance of good approximation algorithms and efficient heuristics for these problems. Then we give an overview of some related work. In the following section we describe Jackson's heuristic, give some basic concepts and facts and derive our estimations on the worst-case performance of Jackson's heuristic. In the final section we present our computational experiments.

II. BASIC CONCEPTS AND DEFINITIONS

We start this section with a more detailed description of Jackson's 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 mini- mum 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 nether 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).

Let S^J be a *J-schedule*, i.e., one obtained by the application of Jackson's heuristic (*J-heuristic*, for short) to the given problem instance. 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_i, t_i) whenever job i starts at its earliest possible starting time, that is, its release time, immediately after the completion of job J_i; here t_i (c_i, respectively) denotes the starting (completion, respectively) time of job J_i.

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 [22], 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 $o(S)$. The *block critical* of S, $B(S)$ is the block containing $o(S)$. Job l is called an *emerging job* in S if $l \in B(S)$ and $q_l < q_{o(S)}$. The latest scheduled emerging job before the overflow job $o(S)$ is called *live*.

The *kernel* of S, $K(S)$ consists of the set of jobs scheduled in S between the live emerging job $l(S)$ and the overflow job $o(S)$, not including $l(S)$ but including $o(S)$ (note that the tail of any job $K(S)$ is no less than of $o(S)$).

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.

If a J-schedule S is not optimal, then the overflow job $o(S)$ must be restarted earlier. In S we look for the overflow job $o(S)$, the kernel $K(S)$and the set of emerging jobs. In order to reduce the maximum job lateness in S ($|S|$), we *apply* an emerging job l for $K(S)$, that is, we *reschedule* l after $K(S)$. Technically, we accomplish this into two steps: first we increase artificially the release time of l, assigning to it a magnitude, no less than the latest job release time in $K(S)$; then apply the J-heuristic to the modified in this way problem instance. Since now the release time of l is no less than that of any job of $K(S)$, and q_l is less than any job tail from $K(S)$, the J-heuristic will give priority to the jobs of $K(S)$ and l will be scheduled after all these jobs. We call the J-schedule obtained from S in this way a weakly complementary to S schedule and denote it by S_l.

III. THE CASES AND CONDITIONS FOR POLYNOMIAL SOLUTION OF THE PROBLEM

In this section we present the conditions leading to the efficient solution of our problem. These conditions are derived by the analysis J-schedule S^J and hence provide an $O(n \log n)$ time decision.

Lemma 1 *If the overflow job $o(S^J)$ is scheduled at its release time, then S^J is optimal.*

Proof. If $o(S^J)$ is scheduled at its release time $r_{o(S)}$ then its starting time $t_{o(S)} = r_{o(S)}$. Hence job $o(S^J)$ starts at its early starting time in S^J and ends at its minimum completion time.

Therefore, $o(S^J)$ can not be rescheduled earlier in S^J, i.e., there is no scheduling $S^{J'}$ such that $\left|S^{J'}\right| < \left|S^J\right|$ and S^J is optimal. □

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

Lemma 2 *If the critical block $B(S^J)$ does not contain an emerging job, then S^J is optimal.*

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

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

If set E is empty then $o(S^J)$ is scheduled in S^J at its release time, therefore by Lemma 1, S^J is optimal. □

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

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

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

Proof. Since the release time r_i each job J_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-algorithm, among all the released by that time moment jobs, will include one with the largest tail. In addition, in S^J, every job is scheduled at its release time r_i or at the completion time of another job. If job $o(S^J)$ is scheduled at time $t = r_{o(S^J)}$, then S^J is optimal by Lemma 1. If $o(S^J)$ is scheduled at the completion time of another job J_k so that job $o(S^J)$ was released before the completion time of that job, then $q_k \geq q_{o(S^J)}$. Hence, there may exist no emerging job and S^J is optimal by Lemma 2. □

Lemma 5 *Let jobs in $J = \left\{J_{i_j}\right\}_{j=1,\ldots,n}$ be ordered by a non-decreasing sequence of their release times. Then S^J is optimal if for any neighboring jobs J_{i_j} and $J_{i_{j+1}}$ $(j = 1, \ldots, n-1)$, $r_{i_{j+1}} \geq r_{i_j} + p_{i_j}$.*

Proof. By the condition in the lemma, every job J_{i_j} $(j = 1, \ldots, n)$ is scheduled in S^J in the interval $\left[r_{i_j}, r_{i_j} + p_{i_j}\right]$. So, the starting time of each job in S^J is $t_j = r_{i_j}$, i. e., each job J_{i_j} begins at its early starting time in S^J. This is true, in particular, for job $o(S^J)$ and therefore, by Lemma 2, S^J is optimal. □

IV. THE NP-HARDNESS RESULT

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, Jackson's 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 $1|r_j|C_{\max}$ ($1||L_{\max}$ and $1|r_j, d_j = d|L_{\max}$) are solvable in an almost linear time $n \log n$ (whereas the general setting $1|r_j, q_j|C_{\max}$ is 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 6 $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\max}$ *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 $B \leq \sum_{i=1}^{n} c_i$. [1] 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 $\sum_{\iota=1}^{n} c_\iota + 1$ as follows.

We have n *partition* jobs $J_1, \ldots, J_n$ released at time $r_1 = 0$ with $p_{J_i} = c_i$, $r_{J_i} = 0$ and $d_{J_i} = \sum_{\iota=1}^{n} c_\iota + 1$, for $i = 1, \ldots, n$. Besides these partition jobs, we have another *separator* job I, released at time $r_2 = B$ with $p_I = 1, r_I = r_2$ 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. 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 $\sum_{\iota=1}^{n} c_\iota + n$ 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 mist be scheduled in the interval $[B, B + 1)$, whereas the latest scheduled partition job must be completed by

[1] In this section n stands exclusively for the number of elements in SUBSET SUM; in the remained part, n denotes the number of jobs

time $\sum_{\iota=1}^{n} c_\iota + 1$. But this may only be possible if the interval $[0, B]$ in schedule S 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 $\sum_{\iota=1}^{n} c_\iota + 1$ plus the total length of the gap within the interval $[0, B]$ and hence that job would not meet its due-date). Now clearly, the processing times of the jobs in set J' form the corresponding solution to SUBSET SUM. □

V. STUDY OF THE RESTRICTED PROBLEM WITH TWO JOB RELEASE TIMES

From here on, we study the version of our general problem in which each job J_i $(i = 1, \ldots, n)$ is released either at the time r_1 or at time r_2. Notice that this version is NP-hard by Theorem 6.

From now on we deal with the sequence of jobs $\{J_{i_j}\}|j = 1, \ldots, m)$ 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 also 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 $\sum_{j=1}^{m} p_{i_j} + r_1 \leq r_2$ then S^J is optimal.*

Proof. Since $\sum_{j=1}^{m} p_{i_j} + r_1 \leq r_2$, J-algorithm in the first m iterations will schedule all the jobs released at time r_1. As the jobs J_{i_j}, $j = 1, ..., m)$ are released simultaneously at the time r_1, 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-algorithm 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). □

Lemma 8 *Let $k < m$ be such that for the first k jobs with the largest tails in schedule S^J the equality $\sum_{l=1}^{k} p_{i_{j_l}} = r_2 - r_1$ holds. Then S^J 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_1, $\sum_{l=1}^{k} p_{i_{j_l}} = r_2 - r_1$ is satisfied:

Case 1: $k = m$. In this case $\sum_{l=1}^{m} p_{i_{j_l}} = r_2 - r_1$ and by Lemma 5, schedule S^J is optimal (with the equality being hold for any tow neighboring jobs).

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

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

Below we let J_i, $i \leq k$, be the earliest arisen job from J_1 during the construction of schedule S^J, such that $t_{J_i} + p_{J_i} > r_2$, and we let q' be the largest tail among the tails of the jobs released at time r_2.

Lemma 9 *If $q_{J_i} \geq q'$, then S^J is optimal.*

Proof. As it is easily seen, because $q_{J_i} \geq q'$, schedule S^J contains no emerging job and hence is optimal by Lemma 2. □

Lemma 10 *If $o(S^J) \in J_1$ then S^J is optimal.*

Proof. For every job $J_i \in J^1$ scheduled in S^J before $o(S^J)$, $q_i \geq q_{o(S^J)}$ (by J-algorithm). Likewise, for each job $J_j \in J^2$ scheduled in S^J before job $o(S^J)$, $q_{J_j} \geq q_{o(S^J)}$, again by J-algorithm. I.e., for any job J_k scheduled in S^J before job $o(S^J)$, $q_{J_k} \geq q_{o(S^J)}$. Hence, S^J does not contain an emerging job and is optimal by Lemma 2. □

Lemma 11 *If $q_{J_i} < q_{J_j}$ and $o(S_\ell) \in J^2$ then S_l is optimal.*

Proof. As $q_{J_i} < q_{J_j}$, then any job scheduled in S_ℓ in the range $\left[r_2, c_{o(S_\ell)}\right]$ is in J_2. Also by Jackson's algorithm any job J_k scheduled before $o(S_l)$ meets that $q_{J_k} \geq q_{o(S_\ell)}$. Therefore the critical block does not contain any emerging job, then by Lemma 2, S_l is optimal. □

VI. CONCLUSION

In this paper, we have analyzed schedules obtained by Jackson's algorithm. This analysis leaded us to the explicit relationships and polynomial solution of these particular cases of our general problem and its spacial case with two job release time, that we have shown to be NP-hard.

As to the directions for the future work, we believe that some of the presented results might be extended for the multiprocessor version of our problem with parallel identical processors.

REFERENCES

[1] P. Bratley, M. Florian and P. Robillard. On sequencing with earliest start times and due–dates with application to computing bounds for ($n/m/G/F_{max}$) problem. *Naval Res. Logist. Quart*. 20, 57–67 (1973)

[2] W. Brinkkotter and P.Brucker. Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments. *J. of Scheduling* 4, 53–64 (2001)

[3] J. Carlier. The one–machine sequencing problem. *European J. of Operations Research*. 11, 42–47 (1982)

[4] J. Carlier and E. Pinson. An Algorithm for Solving Job Shop Problem. *Management Science*, 35, 164-176 (1989)

[5] J. Carlier and E. Pinson. Jakson's pseudo preemptive schedule for the $Pm/r_i, q_i/C_{max}$ problem. *Annals of Operations Research* 83, 41-58, 1998

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

[7] F. Della Croce and V. T'kindt. Improving the preemptive bound for the single machine dynamic maximum lateness problem. *Operations Research Letters* 38 589591 (2010)

[8] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP–completeness. Freeman, San Francisco, 1979.

[9] M.R. Garey, D.S. Johnson, B.B. Simons and R.E. Tarjan. Scheduling unit–time tasks with arbitrary release times and deadlines. *SIAM J. Comput.* 10, 256–269 (1981)

[10] A. Gharbi and M. Labidi. Jackson's Semi-Preemptive Scheduling on a Single Machine. Computers & Operations Research 37, 2082-2088 (2010)

[11] R.L. Graham. E.L. Lawler, J.L. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a servey. *Ann. Discrete Math.* 5 287-326 (1979)

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

[13] Hoogeveen J.A. Minimizing maximum promptness and maximum lateness on a single machine. *Math. Oper. Res* 21, 100-114 (1995)

[14] J.R. Jackson. Schedulig a production line to minimize the maximum tardiness. *Manegement Scince Research Project*, University of California, Los Angeles, CA (1955)

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

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

[17] C.N. Potts. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research* 28, p.1436-1441 (1980)

[18] L. Schrage. Obtaining optimal solutions to resource constrained network scheduling problems, unpublished manuscript (march, 1971)

[19] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM J. Comput.* 12, 294-299 (1983)

[20] B. Simons, M. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM J. Comput.* 18, 690-710 (1989)

[21] N. Vakhania. A better algorithm for sequencing with release and delivery times on identical processors. *Journal of Algorithms* 48, p.273-293 (2003)

[22] N. Vakhania. Single-Machine Scheduling with Release Times and Tails. *Annals of Operations Research*, 129, p.253-271 (2004)

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

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

[25] N.Vakhania. A study of single-machine scheduling problem to maximize throughput. *Journal of Scheduling* Volume 16, Issue 4, pp 395-403 (2013)

[26] N. Vakhania and E.Shchepin. Concurrent operations can be parallelized in scheduling multiprocessor job shop, *J. Scheduling* 5, 227-245 (2002)

[27] N. Vakhania, F. Werner. Minimizing maximum lateness of jobs with naturally bounded job data on a single machine in polynomial time. *Theoretical Computer Science* 501, p. 7281 (2013)

[28] N.Vakhania, D.Perez and L.Carballo. Theoretical Expectation versus Practical Performance of Jackson's Heuristic. *Mathematical Problems in Engineering* Volume 2015, Article ID 484671, 10 pages http://dx.doi.org/10.1155/2015/484671 (2015)