Scheduling jobs with two release times and tails on a single machine

Elisa Chinos,
Nodari Vakhania
Centro de Investigación 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 and its special case when there are only two allowable release times remains to be NP-hard. In this paper we study the latter problem and derive conditions when it can be efficiently solved.

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

I INTRODUCTION

In the broadest sense, the term scheduling can be understood as the assignment of machines or processors over time for a set of jobs or to solve the problem of finding the optimal temporal allocation of certain resources or certain tasks.

Therefore a problem of scheduling there are always three distinct components: the tasks or activities that you want to perform, the resources available for their implementation, and the aims or objectives that you want to achieve and which allow us to identify those which are best among several possible schedules.

Different characteristics of jobs and machines together with different optimality criteria, originate a variety of scheduling problems.

The general scheduling problem studied in this paper is the following: we have n jobs $i(i = 1, ..., n)$ and single machine available at the time 0. Each job i become available at its *release time* or *head* r_i, needs continuous *processing time* p_i on the machine, and need an additional *delivery time* or *tail* q_i, after the completion on the machine (q_i 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 scheduled* S assigns to each job i 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 i, $c_j(S) = t_j(S) + p_j$; the *full completion time* of i in S, $C_j(S) = c_j(S) + q_j$. 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*).

According to the conventional three-field notation introduced by Graham et al. [6] 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 this equivalence, we can use both above formulations interchangeably.

Our generic problem is known to be strongly NP-hard (Garey and Johnson [4]). For exact implicit enumerative algorithms see for instance, McMahon & Florian [10] and Carlier [2]. 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 [9] (see also Schrage [12]). 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 [11] 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 [7] 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 [11].

In a recent work [17], 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 [8]. Garey et al. [5] 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 [13] was proposed an $O(n^2\log n)$ algorithm for the minimization version with two possible job processing times.

For other related criteria, in [14] 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 [16] and [15], respectively, with time complexities $O(n^2\log n)$ and $O(n^3\log n\log p_{\max})$, respectively.

In this paper we consider a special case of the problem $1|r_j,q_j|C_{\max}$ with only two allowable job release times that we abbreviate as $1|\{r_1,r_2\},\{q_i\}|C_{\max}$. This problem was shown to be NP-hard recently in [3]. For this NP-hard problem, we establish different structural properties that lead us to its polynomial-time solution under the corresponding restrictions.

II PRELIMINARIES

In this section we give the basic definitions and notations that we use later. Before that, we give a 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).

We will use σ 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_{opt} for an optimal schedule.

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 i; here t_i (c_i, respectively) denotes the starting (completion, respectively) time of job 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 [13], 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 e is called an *emerging job* in schedule S if $e \in B(S)$ and $q_e < q_{o(S)}$. The latest scheduled emerging job before the overflow job $o(S)$ is called *live* and is denoted by l.

We denote by $E(S)$ the set of all emerging jobs in schedule S.

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 $o(S)$). We 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.

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 . We denote the amount of this forced delay by $\Delta_l = c_l(S) - r(K(S))$. $\Delta_l = 0$ yields that σ is optimal (see also Lemma 8 in [3]):

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

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

In order to reduce the maximum job lateness ($|S|$) 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.

III SOME USEFUL PROPERTIES FOR POLYNOMIAL SOLUTION OF THE PROBLEM

In this section we give some conditions leading to the efficient solution of our generic problem. These conditions are derived as a consequence of the analysis of the J-schedule σ immediately and hence provide an $O(n \log n)$ time decision. Here and later we use some results from reference [3] without stating and prove them (we rather refer to the corresponding results explicatively).

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

Theorem 2 *For a given instance of our generic problem $1|r_j, q_j|C_{\max}$, the initial J-schedule σ 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-algorithm, for each job i scheduled in σ before job $o(\sigma)$, $q_i \geq q_{o(\sigma)}$. Hence, σ does not contain an emerging job and it is optimal (see Lemma 2 from [3]). □

Assume K be 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 3 *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 from kernel $K(\sigma)$ scheduled the last in schedule S cannot complete earlier in schedule S than job $o(S)$ was completed in schedule σ, i.e., $c_{j(S)} \geq c_{o(S^J)}$. At the same time, by the definition of kernel $K(\sigma)$ job j is no less urgent than job $o(\sigma)$, i.e., $q_j \geq q_{o(\sigma)}$. Then $C_{j(S)} \geq C_{o(\sigma)}$ and hence $|S| \geq |\sigma|$. □

Due to Lemma 3 and Lemma 2 from [3], in any schedule better than σ, an emerging job is rescheduled behind kernel $K(\sigma)$.

IV TRACTABLE SPECIAL CASES OF PROBLEM $1|\{r_1, r_2\}, \{q_i\}|C_{\max}$

In [3] we have shown that the version of our general problem with only two job release times, i.e., the problem $1|\{r_1, r_2\}, \{q_i\}|C_{\max}$, is NP-hard. In this section, we establish some useful properties of an optimal solution to this problem that can be verified in time $O(n \log n)$.

IV.1 SPECIAL CASES WHEN $o(\sigma) \in J_2$

Due to Lemma 2, from now on, let us assume that $o(\sigma) \in J_2$.

Observation 4 *For problem $1|\{r_1, r_2\}, \{q_i\}|C_{\max}$, $E(\sigma) \subset J_1$ and for any emerging job $e \neq l$, $q_e \geq q_l$.*

Proof. The first statement follows from the fact that the earliest scheduled job of kernel $K(\sigma)$ in schedule σ belongs to set J_2 which, in turn, follows from the J-heuristic. The second claim also follows from J-heuristic which schedules all jobs released at time r_1 (in particular, the jobs from $E(\sigma)$) in the non-increasing order of their tails (in particular, up to time r_2 before the jobs of kernel $K(S^J)$), whereas job l is the latest scheduled one from set $E(\sigma)$ in schedule σ. □

Recall that, for a given J-schedule S, the complementary schedule S_l always contains a gap before jobs of kernel $K(S)$, i.e., the earliest scheduled job of $K(S)$ starts at its release time in schedule S_l. In general, for any emerging job $e \in E(S)$, the complementary schedule S_e 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 r_2, and before and after job e (particularly that of the jobs of $K(\sigma)$) is the same in both schedules σ and σ_e.*

Proof. Note that since all jobs of kernel $K(\sigma)$ and the jobs scheduled after this kernel in σ were released by time r_2, J-algorithm 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_e, i.e., the jobs before job e and after that job (also released by time r_2), in particular, ones of kernel $K(\sigma)$ will be included in the same order in both schedules. □

Let now $J[e]$ be the set of all jobs j scheduled after r_2 in σ such that $q_e < q_j < q_{o(\sigma)}$. It follows that all jobs from set $J[e]$ are scheduled immediately after kernel $K(\sigma)$ and before job e in σ_e. Besides,

Observation 6 $J[e] \subset J_2$.

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

Proposition 7 *Without loss of generality, it might be assumed that all the jobs j with $q_j = q_e$ scheduled after kernel $K(\sigma)$ in σ are included behind job e in complementary schedule σ_e.*

Observation 8 *In schedule σ_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 σ, J-algorithm schedules all the jobs in the non-increasing order of their tails behind time r_2 (as all of them are released by that time moment). While constructing schedule σ_e, the same set of jobs plus job e are available behind time r_2. In particular, it is easy to see that if $p_e \geq \Delta_l$, all jobs of kernel $K(\sigma)$ and those from set $J[e]$ will be left-shifted by Δ_l, and if $p_e < \Delta_l$ 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 9 *1. If $p_e \geq \Delta_l$, then σ_e has a gap of length $p_e - \Delta_l$.*

2. *If $p_e < \Delta_l$, then σ_e has not a gap.*

Proof. First note that Δ_l is now $c_l(\sigma) - r_2$. By the definition of σ_e, no job $j \in J_1$ with $q_j < q_e$ scheduled after $K(\sigma)$ in σ will be scheduled before $K(\sigma)$ in σ_e. If $p_e \geq \Delta_l$, in schedule σ_e, the jobs of kernel $K(\sigma)$ and of set $J[e]$ will have the left-shift of length Δ_l (which is the maximum possible by Observation 1). Then σ_e will have a gap of length $p_e - \Delta_l$. If $p_e < \Delta_l$, the jobs of kernel $K(\sigma)$ and those of set $J[e]$ will have a left-shift of length $\Delta_l - p_e$, hence σ_e will have no gap. □

Due to Lemma (9) we define the gap δ_e in schedule σ_e as follows:

$$\delta_e = \begin{cases} 0 & if \quad p_e < \Delta_l \\ p_e - \Delta_l & if \quad p_e \geq \Delta_l \end{cases}$$

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

Lemma 10 *The full completion time of job e in σ_e,*

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

Proof. By the definition of the set $J[e]$, job e will be scheduled in σ_e after all jobs of $J[e]$ and before of all jobs i with $q_i = q_e$ (see proposition (7)). By the observation (1), Δ_l is the maximum possible left-shift for the jobs of $K(\sigma)$ in σ_e. The full completion time of job e in σ_e will depend on the amount of this left-shift. Consider the following two cases. If $p_e \geq \Delta_l$, then the jobs of $K(\sigma)$ and those of set $J[e]$ will be left-shifted by the amount Δ_l in schedule σ_e. Recall that the last job of kernel $K(\sigma)$ scheduled in σ is $o(\sigma)$. Then,

$$C_{e(\sigma_e)} = (c_{o(\sigma)} + \sum_{j \in J[e]} p_j) - \Delta_l + p_e + q_e.$$

Let $\delta_e = p_e - \Delta_l$, then $p_e = \delta_e + \Delta_l$. Therefore

$$\begin{aligned} C_{e(\sigma_e)} &= c_{o(\sigma)} + \sum_{j \in J[e]} p_j - \Delta_l + \delta_e + \Delta_l + q_e \\ &= c_{o(\sigma)} + \sum_{j \in J[e]} p_j + \delta_e + q_e. \end{aligned}$$

In this case, σ_e has a gap of length δ_e.

If $p_e < \Delta_l$, then by the definition of set $J[e]$ and the construction of schedule σ_e, the jobs of kernel $K(\sigma)$ and those of set $J[e]$ will be left-shifted by amount p_e in schedule σ_e. Therefore,

$$\begin{aligned} C_{e(\sigma_e)} &= (c_{o(\sigma)} + \sum_{j \in J[e]} p_j) - p_e + p_e + q_e \\ &= c_{o(\sigma)} + \sum_{j \in J[e]} p_j + q_e. \end{aligned}$$

Note that in this case σ_e has no gap. □

Consider the following inequality:

$$C_{e(\sigma_e)} = c_{o(\sigma)} + \sum_{j \in J[e]} p_j + \delta_e + q_e \geq C_{o(\sigma)}, \qquad (1)$$

Lemma 11 *If $p_e \geq \Delta_l$ 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 $p_e \geq \Delta_l$, σ_e has a gap by Lemma (9) and kernel $K(\sigma)$ starts at its early starting time $r(K(\sigma)) = r_2$ in schedule σ_e. By inequality (1), $C_{e(\sigma_e)} \geq C_{o(\sigma)}$; 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 12 *If $p_e \geq \Delta_l$ for every $e \in E(\sigma)$ and the inequality (1) is satisfied for job e, then σ is optimal.*

Proof. Recall that if schedule σ 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 σ_e is a schedule with a better (smaller) makespan than schedule σ. Then similarly, it is straightforward to verify that, by the condition of the lemma, inequality (1) yields $C_{e(\sigma_e)} \geq C_{o(\sigma)}$, for every $e \in E(\sigma)$. Then by Lemma 11, schedule σ 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 11, l is scheduled in S_{opt} before $K(S_{opt})$. Let k be the last job scheduled of $J[l]$ in σ (recall that $J[l]$ is the set of all jobs j scheduled after r_2 in σ such that $q_l < q_j < q_{o(\sigma)}$). By J-algorithm, 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-algorithm schedules job l after job k in σ_l. In addition, the starting time of job l must be the completion time of job k in σ_l, i. e., $t_{l(\sigma_l)} = c_{k(\sigma_l)}$ (by proposition 7 and because by J-algorithm, for any job i scheduled after of k in σ, $q_i < q_k$ and therefore $q_i \leq q_l$, by the definition of set $J[l]$).

It is easy to see that for all $e \in E(\sigma, l)$, $\delta_e \geq \delta_l$ (because $p_e \geq p_l$ and by the definition of δ_e). Besides, job k is scheduled after jobs of kernel $K(\sigma)$ in σ_e and job e can be scheduled before or after job k in σ_e. We respectively distinguish the following two cases. If job e is scheduled in σ_e after of job k, the starting time of job e in schedule σ_e is equal to the starting time of job l in schedule σ_l, i. e., $t_{e(\sigma_e)} = t_{l(\sigma_l)}$ since for any job i scheduled after job l in σ_l, $q_i \leq q_l$ and $q_i < q_e$ since $q_e \geq q_l$. As the inequality (1) is satisfied for l, it is also satisfied for job e because $\delta_e \geq \delta_l$ and $q_e \geq q_l$. By Lemma 11, e is scheduled before $K(\sigma)$ in S_{opt}.

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

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

IV.2 A FEW MORE PROPERTIES FOR EQUAL-LENGTH JOBS IN SET J_1

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

Among all the jobs scheduled after jobs in kernel $K(\sigma)$ and set $J[e]$ in schedule σ, let k be the job with the greatest full time completion in σ. Recall that the left-hand side in inequality (1) is the full completion time of job e in schedule σ_e. If $C_{e(\sigma_e)}$ is not greater than $C_{o(\sigma)}$ then the full completion time of job k in σ_e may be greater than the full completion time of job $o(\sigma)$ in σ.

We define set $J(e,k) = \{i \mid q_k \leq q_i \leq q_e\}$.

Lemma 14 *The full completion time of job k in the schedule σ_e is:*

$$C_{k(\sigma_e)} = c_{o(S)} + \sum_{j \in J[e]} p_j + \delta_e + \sum_{i \in J(e,k)} p_i + p_k + q_k.$$

Proof. Is analogous to that of Lemma (10). □

Consider the following inequality:

$$c_{o(S)} + \sum_{j \in J[e]} p_j + \delta_e + \sum_{i \in J(e,k)} p_i + p_k + q_k \geq C_{o(S)} \quad (2)$$

Lemma 15 *For problem $1\left|\{r_1, r_2\}, q_j, p_{r_1,j} = p\right| C_{max}$, in any schedule σ_e there occurs a gap before kernel $K(\sigma)$. Moreover, $\delta_e = \delta_l$ for every job $e \in E(\sigma) \setminus l$.*

Proof. We have $p_e > \Delta_l$ since $p_e = p_l = p$ for all $e \in E(\sigma) \setminus \{l\}$. By Lemma 9, schedule σ_e has a gap of length $p_e - \Delta_l$ and $\delta_e = \delta_l$ for all $e \in E(\sigma) \setminus \{l\}$. □

Lemma 16 *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 $C_{k(\sigma_l)} \geq C_{o(\sigma)}$; hence, $|\sigma_l| \geq |\sigma|$. As the full completion time of k in σ is less than the full completion time of the overflow job $o(\sigma)$ (otherwise, job k would be the overflow job in σ) and $C_{k(\sigma_l)} \geq C_{o(\sigma)}$, job l cannot be scheduled after kernel $K(\sigma)$ in schedule S_{opt}. By Lemma (15), $\delta_e = \delta_l$ for every emerging job e. Then the starting time of job k in schedule σ_e is the same as that in σ_l. Therefore $C_{k(\sigma_e)} \geq C_{o(\sigma)}$ and $|\sigma_e| \geq |\sigma|$ 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}. □

Theorem 17 *For problem $1\left|\{r_1, r_2\}, q_j, p_{r_1,j} = p\right| C_{max}$, schedule σ is optimal, if:*

1. *The inequality (1) is satisfied for job l in schedule σ_l.*
2. *The inequality (2) is satisfied for job k in schedule σ_l.*

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 (12) σ is optimal.

As to the second claim, by Lemma (16) every job $e \in E(\sigma) \setminus \{l\}$ must be scheduled before kernel $K(\sigma)$ in S_{opt}, hence σ is optimal. □

V CONCLUSION

The presented properties of problem $1|\{r_1, r_2\}, \{q_i\}|C_{\max}$ have, from one hand, the theoretical value as they lead us to optimal solution in a low degree polynomial time. From

the other hand, these properties have also the practical importance as they can be efficiently used in enumerative algorithms for this NP-hard problem.

As to the directions for the future work, we believe that the presented approach 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] J. Carlier. The one–machine sequencing problem. *European J. of Operations Research*. 11, 42–47 (1982)

[3] E. Chinos and N. Vakhania. Polynomially Solvable and NP-hard Special Cases for Scheduling with Heads and Tails. *RECENT ADVANCES in MATHEMATICS and COMPUTATIONAL SCIENCE Proceedings of the 4th International Conference on Mathematical, Computational and Statistical Sciences (MCSS '16)*, p.141-145 (2016)

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

[5] 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)

[6] 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)

[7] 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)

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

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

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

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

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

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

[14] 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)

[15] 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)

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

[17] 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)