Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 484671, 10 pages
http://dx.doi.org/10.1155/2015/484671

Research Article

Theoretical Expectation versus Practical Performance of Jackson's Heuristic

Nodari Vakhania, Dante Pérez, and Lester Carballo

Facultad de Ciencias, UAEM, Avenida Universidad 1001, 62210 Cuernavaca, MOR, Mexico

Correspondence should be addressed to Nodari Vakhania; nodari@uaem.mx

Received 19 January 2015; Revised 14 May 2015; Accepted 18 May 2015

Academic Editor: Ben T. Nohara

A basic 2-approximation heuristic was suggested by Jackson in early 50s last century for scheduling jobs with release times and due dates to minimize the maximum job lateness. The theoretical worst-case bound of 2 helps a little in practice, when the solution quality is important. The quality of the solution delivered by Jackson's heuristic is closely related to the maximum job processing time $p_{\max}$ that occurs in a given problem instance and with the resultant interference with other jobs that such a long job may cause. We use the relationship of $p_{\max}$ with the optimal objective value to obtain more accurate approximation ratio, which may drastically outperform the earlier known worst-case ratio of 2. This is proved, in practice, by our computational experiments.

1. Introduction

One of the oldest and commonly used (online) heuristics in scheduling theory is that of Jackson [1] (see also Schrage [2]). It was first suggested for scheduling jobs with release times and due dates on a single machine to minimize the maximum job lateness. In general, variations of Jackson's heuristic are widely used to construct feasible solutions for scheduling jobs with release and delivery times on a single machine or on a group of parallel machines. Besides, Jackson's heuristic is efficiently used in the solution of more complicated shop scheduling problems including job-shop scheduling, in which the original problem occurs as an auxiliary one and is applied to obtain lower estimations in implicit enumeration algorithms (although even this latter problem is strongly NP-hard, see Garey and Johnson [3]). At the same time, it is known that, in the worst-case, Jackson's heuristic will deliver a solution which is twice worse than an optimal one.

The latter worst-case bound might be too rough in practice, when the solution quality is important; that is, solutions with the objective value better than twice the optimal objective value are required. In addition, the solutions may need to be created online (any possible offline modification of the heuristic that may lead to a better performance would be of not much use). In this situation, the online performance measure is essentially important.

As we will see, the quality of the solution delivered by the heuristic is essentially related to the maximum job processing time $p_{\max}$ that may occur in a given problem instance. In particular, the interference of a "long" nonurgent job with the following scheduled urgent jobs affects the solution quality.

We express $p_{\max}$ as a fraction the optimal objective value and derive a much more accurate approximation ratio than 2. In some applications, this kind of relationship can be predicted with a good accuracy. For instance, consider large-scale production process where the processing requirement of any individual job is small relative to an estimated (shortest possible) overall production time T of all the products (due to a large number of products and the number of operations required to produce each product). If this kind of prediction is not possible, by a single application of Jackson's heuristic, we obtain a strong lower bound on the optimal objective value and represent $p_{\max}$ as its fraction κ (instead of representing it as a fraction of an unknown optimal objective value). Then we give an explicit expression of the heuristic's approximation ratio in terms of that fraction. In particular, Jackson's heuristic will always deliver a solution within a factor of $1 + 1/\kappa$ of

the optimum. We further refine $p_{\max}$ to a smaller more accurate magnitude and use it instead of $p_{\max}$ in our second stronger estimation.

Our estimations may drastically outperform the earlier known worst-case ratio of 2. This is proved, in practice, by the computational experiments. From 200 randomly generated problem instances, more than half of the instances were solved optimally by Jackson's heuristic, as no above interference with a long job has occurred. For the rest of the instances, the interference was insignificant, so that most of them were solved within a factor of 1.009 of the optimum objective value, whereas the worst approximation ratio was less than 1.03. According to the experimental results, our lower bounds turn out to be quite strong, in practice.

The paper is organized as follows. 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.

1.1. Heuristics and Combinatorial Optimization Problems. *Combinatorial optimization* (CO) problems constitute a significant class of practical problems with a discrete nature. They have emerged in late 40s of 20th century. With a rapid growth of the industry, the new demands in the optimal solution of the newly emerging resource management and distribution problems have arisen. For the development of effective solution methods, these problems were formalized and addressed mathematically.

A CO problem is characterized by a finite set of the so-called *feasible solutions*, defined by a given set of restrictions, and an objective function for these feasible solutions, which typically needs to be optimized, that is, minimized or maximized: the problem is to find an *optimal solution*, that is, one minimizing the objective function. Typically, the number of feasible solutions is finite. Thus theoretically, finding an optimal solution is trivial: just enumerate all the feasible solutions calculating for each of them the value of the objective function and select any one with the optimal objective value. However, this brutal enumeration of all feasible solutions may be impossible in practice. Even for problems with a moderate size (say 30 cities for a classical traveling salesman problem or 10 jobs on 10 machines in job-shop scheduling problem), such a complete enumeration may take hundreds of centuries on the modern computers. Moreover, this situation will not be changed if in the future, much faster computers will be developed.

The CO problems are partitioned into two basic types, type P, which are polynomially solvable ones, and NP-hard problems. Intuitively, there exist efficient (polynomial in the *size* of the problem) solution methods or algorithms for the problems from the first class, whereas no such algorithms exist for the problems of the second class (informally, the size of the problem is the number of bits necessary to represent the problem data/parameters in a computer memory). Furthermore, all NP-hard problems, ones from the second class, have a similar *computational complexity*, in the sense that a polynomial-time algorithm for any of them would yield a polynomial time algorithm for any other problem from this class. At the same time, it is believed that it is very unlikely that an NP-hard problem can be solved in polynomial time.

Greedy (heuristic) algorithms are efficient polynomial-time algorithms that create a feasible schedule. Such algorithms, typically, work on n (external) iterations, where n is the number of objects in the given problem. The *size* of the problem (i.e., the number of bits necessary to represent the problem data/parameters in a computer memory) is a polynomial in n. Hence, the number of iterations in a greedy algorithm is also polynomial in the size of the problem. Such an algorithm creates a complete destiny feasible solution iteratively, extending the current partial solution by some yet unconsidered object at each iteration. In this way, the search space is reduced to a single possible extension at each iteration, from all the theoretically possible potential extensions. This type of "rough" reduction of the whole solution space may lead us to the loss of an optimal or near-optimal solution.

A greedy/heuristic algorithm may create an optimal solution for a problem in class P (though not any problem in class P may optimally be solved by a heuristic method). However, this is not the case for an NP-hard problem; that is, no greedy or heuristic algorithm can solve optimally an NP-hard problem (unless $P = \mathrm{NP}$, which is very unlikely). Since the majority of CO problems are NP-hard, a compromise accepting a solution worse than an optimal one is hence unavoidable. On this way, it is natural and also practical to think about the design and analysis of polynomial-time *approximation algorithms*, that is, ones which deliver a solution with a guaranteed deviation from an optimal one in polynomial time. The *performance ratio* of an approximation algorithm A is the ratio of the value of the objective function delivered by A to the optimal value. A *κ-approximation algorithm* is one with the worst-case performance ratio κ. Since the simplest polynomial-time algorithms are greedy, a greedy algorithm is a simplest approximation algorithm.

1.1.1. Scheduling Problems. 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 given objective criteria.

A basic scheduling problem that we consider in this paper is as follows: n jobs have to be scheduled on a single machine. Each job j becomes available at its *release time* r_j. A released job can be assigned to the machine that has to process job j for p_j time units. The machine can handle at most one job at a time. Once it completes j this job still needs a (constant) *delivery time* q_j for its *full completion* (the jobs are delivered by an independent unit and this takes no machine time). All above parameters are integers. Our 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. [4] 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 are given. The problem has an equivalent formulation $1|r_j|L_{\max}$ in which delivery times are interchanged by *due dates* and the maximum job *lateness* $L_{\max}$, that is, the difference between the job completion time and its due date, is minimized (due date d_j of job j is the desirable time for the completion of job j; there occurs a penalty whenever j is completed after time moment d_j).

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; that is, 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. [5] for more details).

Because of the above equivalence, we will use both above formulations interchangeably. (As noted briefly earlier, the version with delivery times naturally occurs in implicit enumeration algorithms for job-shop scheduling problem and is used for the calculation of lower bounds.)

1.2. Overview. 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 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 $O(n \log n)$.

A number of efficient algorithms are variations of Jackson's heuristic. For instance, Potts [6] has proposed a modification of this heuristic with 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 same 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 [6]. Jackson's heuristic can be efficiently used for the solution of shop scheduling problems. Using Jackson's heuristic as a schedule generator, McMahon and Florian [8] and Carlier [9] have proposed efficient enumerative algorithms for $1|r_j, q_j|C_{\max}$. Grabowski et al. [10] use the heuristic for the obtention of an initial solution in another enumerative algorithm for the same problem. Garey et al. [11] have applied the same heuristic in an $O(n \log n)$ algorithm for the feasibility version of this problem with equal-length jobs (in the feasibility version job due-dates are replaced by deadlines and a schedule in which all jobs are complete by their deadlines is looked for). Again using Jackson's heuristic as a schedule generator, other polynomial-time direct combinatorial algorithms were described. In [12] was proposed an $O(n^2 \log n)$ algorithm for the minimization version of the latter problem with two possible job processing times and in [13] 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 [14] and [15], respectively, with time complexities $O(n^2 \log n)$ and $O(n^3 \log n \log p_{\max})$, respectively.

Jackson's heuristic has been used in multiprocessor scheduling problems as well. 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 [16] and Simons and Warmuth [17], respectively. Using the same heuristic as a schedule generator in [18] 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 heuristic has been successfully used for the obtainment of lower bounds in 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 [9], Carlier and Pinson [19], and Brinkkotter and Brucker [20] and more recent works of Gharbi and Labidi [21] and Croce and T'kindt [22]. Carlier and Pinson [23] 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 [24]). Jackson's heuristic can be useful for parallelizing the computations in scheduling job-shop Perregaard and Clausen [25] and also for the parallel batch scheduling problems with release times Condotta et al. [26].

2. Theoretical Estimations of Heuristics Performance

We start this section with 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 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 so far assigned job to the machine or the minimum release time of a yet unassigned job, whichever is more (since no job can be started before the machine gets idle, nether it can be started 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 avoiding 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, alternatively, with the largest delivery time).

Let σ be the schedule obtained by the application of Jackson's heuristic (*J-heuristic, for short*) to the originally given problem instance. Schedule, σ, and, in general, any Jackson's schedule S (*J-schedule*, for short), that is, one constructed by J-heuristic, 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_i) whenever job i starts at its earliest possible starting time, that is, its release time, immediately after the completion of job j; here t_j (c_j, resp.) denotes the starting (completion, resp.) 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 [18], will help us to expose these properties.

Given a J-schedule S, let i be a job that realizes the maximum job lateness in S; that is, $L_i(S) = \max_j\{L_j(S)\}$. Let, further, B be the block in S that contains job i. Among all the jobs in B with this property, the latest scheduled one is called an *overflow job* in S (we just note that not necessarily this job ends block B).

A *kernel* in S is a maximal (consecutive) job sequence ending with an overflow job o such that no job from this sequence has a due date more than d_o. For a kernel K, we let $r(K) = \min_{i\in K}\{r_i\}$.

It follows that every kernel is contained in some block in S, and the number of kernels in S equals the number of the overflow jobs in it. Furthermore, since any kernel belongs to a single block, it may contain no gap.

The following lemmas are used in the proof of Theorem 6. A statement similar to Lemma 1 can be found in [27] and Lemma 3 in [18]. Lemma 5 is obtained as a consequence of these two lemmas, though the related result has been known earlier. For the sake of completeness of our presentation, we give all our claims with proofs.

Lemma 1. *The maximum job lateness (the makespan) of a kernel K cannot be reduced if the earliest scheduled job in K starts at time $r(K)$. Hence, if a J-schedule S contains a kernel with this property, then it is optimal.*

Proof. Recall that all jobs in K are no less urgent than the overflow job o and that jobs in K form a tight sequence (i.e., without any gap). Then since the earliest job in K starts at its release time, no reordering of jobs in K can reduce the current maximum lateness, which is $L_o(S)$. Hence, there is no feasible schedule S' with $L(S') < L_o(S)$; that is, S is optimal. □

Thus σ is already optimal if the condition in Lemma 1 holds. Otherwise, there must exist a job less urgent than o, scheduled before all jobs of kernel K that delays jobs in K (and the overflow job o). By rescheduling such a job to a later time moment the jobs in kernel K can be restarted earlier. We need some extra definitions to define this operation formally.

Suppose job i precedes job j in ED-schedule S. We will say that *i pushes j* in S if ED-heuristic will reschedule job j earlier whenever i is forced to be scheduled behind j.

Since the earliest scheduled job of kernel K does not start at its release time (Lemma 1), it is immediately preceded and pushed by a job l with $d_l > d_o$. In general, we may have more than one such a job scheduled before kernel K in block B (one containing K). We call such a job an *emerging job* for K, and we call the latest scheduled one (job l above) the *live* emerging job.

From the above definition and Lemma 1 we immediately obtain the following corollary.

Corollary 2. *If S contains a kernel which has no live emerging job, then it is optimal.*

We illustrate the above introduced definitions on a problem instance of $1|r_j, q_j|C_{\max}$. The corresponding J-schedule is depicted in Figure 1(a). In that instance, we have 11 jobs, job 1 with $p_1 = 100$, $r_1 = 0$, and $q_1 = 0$. All the rest of the jobs are released at time moment 10 and have the equal processing time 1 and the delivery time 100. These data completely define our problem instance.

Consider the initial J-schedule σ of Figure 1(a) consisting of a single block. In that schedule, jobs are included in the increasing order of their indexes. The earliest scheduled job 1 is the live emerging job which is followed by jobs 2–11 scheduled in this order (note that, for the technical reasons, the scaling on the vertical and horizontal axes is different). It is easy to see that the latter jobs form the kernel K in schedule σ. Indeed, all the 11 jobs belong to the same block, job 1 pushes the following jobs, and its delivery time is less than that of these pushed jobs. Hence, job 1 is the live emerging job in schedule σ. The overflow job is job 11 since it realizes the value of the maximum full completion time (the makespan) in schedule σ which is $110 + 100 = 210$. Therefore, jobs 2–11 form the kernel in σ.

Note that the condition in Lemma 1 is not satisfied for schedule σ. Indeed, its kernel K starts at time 100 which is more than $r(K) = 10$. Furthermore, the condition of Corollary 2 is also not satisfied for schedule σ and it is not optimal. The optimal schedule S^* with makespan 120 is depicted in Figure 1(b), in which the live emerging job 1 is rescheduled behind all kernel jobs.

Below we use T^S for the makespan (maximum full job completion time) of J-schedule S and T^* ($L^*_{\max}$, resp.) for the optimum makespan (lateness, resp.).

Lemma 3. *Consider $T^\sigma - T^* < p_l$ ($L^\sigma_{\max} - L^*_{\max} < p_l$), where l is the live emerging job for kernel $K \in \sigma$.*

Proof. We need to show that the delay imposed by job l for the jobs in kernel K in schedule σ is less than p_l. Indeed, σ is a J-schedule. Hence, no job in K could have been released by the time when job l was started in σ, as otherwise J-heuristic would have included the former job instead of l. At the same

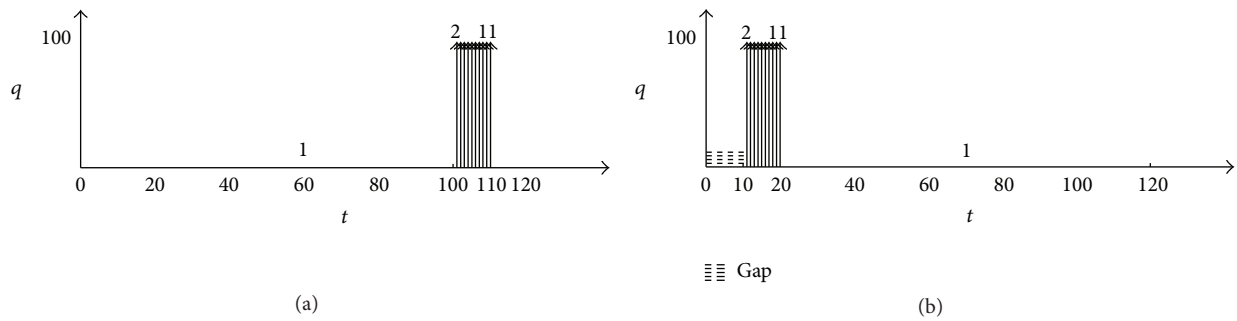

Figure 1: First instance. On (a) the J-schedule σ and on (b) the optimal schedule S^*.

time, the earliest job from K is scheduled immediately after job l in σ. Then the difference between the starting time of the former job and time moment $r(K)$ is less than p. Now our claim follows from Lemma 1. □

For our problem instance and the corresponding schedule σ, the above bound is almost reached. Indeed, $T^\sigma - T^* = 210 - 120 = 90$, whereas $p_l = 100$ ($l = 1$).

As we have mentioned in Section 1.2, Jackson's heuristic's preemptive version (which gives an optimal solution to $1|r_j, q_j, pmtn|C_{\max}$) gives a lower bound for $1|r_j, q_j|C_{\max}$; that is, an optimal solution of the preemptive version $1|r_j, q_j, pmtn|C_{\max}$ is a lower bound for the non-preemptive case $1|r_j, q_j|C_{\max}$. By going deeper into the structure of Jackson's preemptive schedules, Croce and T'kindt [22] have proposed another (a more "complete") lower bound, which, in practice, also turns out to be more efficient than the above lower bound yielded by an optimal preemptive solution. The lower bound proposed by Gharbi and Labidi [21] is based on the concept of the so-called semipreemptive schedules, derived from the observation that in an optimal nonpreemptive schedule a part of some jobs is to be scheduled within a certain time interval. This yields to the semipreemptive schedules, for which stronger lower bounds can be derived.

Unlike the above lower bounds, Lemma 3 implicitly defines a lower bound of $T^\sigma - p_l$ derived from the solution of the nonpreemptive Jackson's heuristic. This lower bound can further be strengthen using the following concept. Let the delay for kernel $K \in \sigma$, $\delta(K, l)$ be $c_l - r(K)$ (l (o, resp.) stand again for the live emerging (overflow, resp.) job for kernel K).

Lemma 4. *$L^* = T^\sigma - \delta(K, l)$ ($L_o(\sigma) - \delta(K, l)$, resp.) is a lower bound on the optimal job makespan T^* (lateness $L^*_{\max}$, resp.).*

The proof is similar to that of Lemma 3, with an extra observation that the delay for the earliest scheduled job of kernel K is defined more accurately by $\delta(K, l)$.

Observe that $\delta(K, l) < p_l$. In fact, $\delta(K, l)$ can be drastically smaller than p_l. For instance, if in our problem instance from Figure 1 $r(K)$ were 90 (instead of 10) then $\delta(K, l) = 100 - 90 = 10$. In general, observe that the smaller is $\delta(K, l)$ (the more is $p_l - \delta(K, l)$) the more essential is the difference between the lower bounds of Lemmas 4 and 3.

Now we can easily derive a known performance ratio 2 of J-heuristic for version $1|r_j, q_j|C_{\max}$ (we note that the estimation of the approximation for the version with due dates with the objective to minimize maximum lateness is less appropriate: for instance, the optimum lateness might be negative).

Lemma 5. *J-heuristic gives a 2-approximate solution for $1|r_j, q_j|C_{\max}$; that is, $T^\sigma/T^* < 2$.*

Proof. If there exists no live emerging job for $K \in \sigma$ then σ is optimal by Corollary 2. Suppose l exists; clearly, $p_l < T^*$ (as job l has to be scheduled in S^* and there is at least one more (kernel) job in it). Then by Lemma 3,

$$\frac{T^\sigma}{T^*} < \frac{(T^* + p_l)}{T^*} = 1 + \frac{p_l}{T^*} < 1 + 1 = 2. \quad (1)$$

□

For the purpose of the estimation of the approximation given by Jackson's heuristic, we express p_l as a fraction of an optimal objective value T^* ($L^*_{\max}$). Alternatively, instead of the optimal objective value we may use its lower bound L^* from Lemma 4 (as T^* may not be known). Let $\kappa > 1$ be such that $p_l \le T^*/\kappa$; that is, $\kappa \le T^*/p_l$. Since L^* is a lower bound on T^* ($L^* \le T^*$), we let $\kappa = L^*/p_l$, and thus we have that $\kappa \le T^*/p_l$; that is, $\kappa = L^*/p_l$ is a valid assignment. Then note that for any problem instance κ can be obtained in time $O(n \log n)$.

In the following two theorems l is the live emerging job for kernel $K \in \sigma$, as before.

Theorem 6. *Consider $T^\sigma/T^* < 1 + 1/\kappa$, for any $\kappa \le T^*/p_l$.*

Proof. By Lemma 3,

$$\frac{T^\sigma}{T^*} < \frac{(T^* + p_l)}{T^*} = 1 + \frac{p_l}{T^*} \le 1 + \frac{1}{\kappa}. \quad (2)$$

□

Similarly as in Lemma 4, we can strengthen Theorem 6 replacing p_l by $\delta(K, l)$ redefining, respectively, κ; that is, we now allow $\kappa \le T^*/\delta(K, l)$ (where we may let $\kappa = L^*/\delta(K, l)$).

Theorem 7. *Consider $T^\sigma/T^* \le 1 + 1/\kappa$, for any $\kappa \le T^*/\delta(K, l)$.*

Proof. The proof is saimilar to the proof of Theorem 6 with the only difference that the strong inequality in $T^\sigma/T^* <$

$(T^* + p_l)/T^*$ is replaced by $T^\sigma/T^* \leq (T^* + \delta(K,l))/T^*$ as now $T^\sigma \leq T^* + \delta(K,l)$. □

Note that if we let $\kappa = L^*/p_l$ and $\kappa = L^*/\delta(K,l)$, respectively, in Theorems 6 and 7, respectively, and we replace T^* with the lower bound L^*, we obtain valid inequalities $T^\sigma/L^* < 1 + 1/\kappa$, for any $\kappa \leq L^*/p_l$ and $T^\sigma/L^* \leq 1 + 1/\kappa$, for any $\kappa \leq L^*/\delta(K,l)$, respectively.

To illustrate the above obtained results, let us consider another modification of our problem instance of Figure 1. In that modified instance the emerging job remains longer than the kernel jobs, although the difference between the processing times is not as drastic as in the previous instance (such an instance characterizes better an "average problem instance"). We have a long emerging job 1 with $p_1 = 10$, $r_1 = 0$, and $q_1 = 0$, and the processing time of the rest of the jobs is again 1. Latter jobs are released at time 5 and also have the same delivery times as in the first instance. The J-schedule σ with makespan 120 is depicted in Figure 2(a). The lower bound on the optimum makespan defined by Lemma 3 is hence $120 - 10 = 110$, whereas Lemma 4 defines a stronger lower bound $120 - 5 = 115$, since $\delta(K,1) = 5$. The makespan of the optimal schedule depicted in Figure 2(b) is the same as this lower bound.

The approximation provided by Jackson's heuristic for this problem instance can be obtained from Theorems 6 and 7. Based on Theorem 6, we use the lower bound 115 on T^* and obtain a valid $\kappa = T^*/p_l = 115/10 = 11.5$ and the resultant approximation ratio $1 + 1/11.5$. Using Theorem 7 and the fact that $\delta(K,1) = 5$ with the lower bound 115, we obtain another valid $\kappa = T^*/\delta(K,1) = 115/5 = 23$ and the approximation ratio $1 + 1/23$. For this instance, we can also obtain the approximation ratio directly $120/115 = 1 + 1/23$, which coincides with the estimation of Theorem 7.

Observe that for our second (average) problem instance, Jackson's heuristic gave an almost optimal solution, and the resultant approximation ratio coincides with the estimation of Theorem 7. This encouraging observation is completely supported and even outperformed by our computational experiments discussed in Section 3.

3. On Heuristic's Practical Behavior

Recall that in our first problem instance from Figure 1 we had a "huge" live emerging job essentially contributing to the makespan of σ. As a result, Jackson's heuristic has created a schedule with an almost worst possible performance ratio (see Lemma 5). Two distinct facts were decisive in such a poor performance of the J-heuristic. First, it the processing time of the live emerging job (which was too large); second, it is also large $\delta(K,l)$ "close" to p_l. We have carried out computational experiments aiming to verify how often, in practice, both of these two events occur. The results were more than encouraging showing that it is highly unlikely that both of these events may take place, as we describe a bit later in this section. As an example, for the second modified problem instance with $r(K) = 90$ from the previous section, the second event did not occur (as $\delta(K,l)$ was 10, instead of 90, a value, relatively small compared to the optimal objective value). As a result, Jackson's heuristic has provided a good approximation. Our third problem instance from Figure 2(a) reflects typical characteristics of an "average" problem instance.

Our study has shown that in real-life scenarios, where the processing requirement of an individual (live emerging) job can approximately be estimated as a fraction of the expected total work time, the quality of the solution that will deliver Jackson's heuristic can be predicted in terms of that fraction (Theorems 6 and 7) without actually running the algorithm and can be significantly better than the known worst-case bound of 2. Alternatively, Lemmas 3 and 4 provide the lower bounds on the expected total work time, and the above fraction can directly be derived.

3.1. The Computational Experiments. We have implemented Jackson's heuristic (the version $1|r_j|L_{\max}$) in Java using the development environment Eclipse IDE for Java Developers (version Luna Service Release 1 (4.4.1)) under Windows 8.1 operative system for 64 bits and have used a laptop with Intel Core i7 (2.4 GHz) and 8 GB of RAM DDR3 to run the code. The inputs in our main program are plain texts with job data that we have generated randomly. The program for the generation of our instances was constructed under the same development environment as our main program. The job parameters (the release times, the processing times, and the due dates) were generated randomly, somewhat similar as in [9, 21], as follows. For job release times and due dates a random number was generated with the rnd() function in Java, with an open range $(0, 50n)$, where n is the number of jobs in each instance. The processing times were generated from the interval $[1, 50]$ (as in [9, 21]) and also from the interval $[1, 100]$.

For deeper analysis of the created solutions, we have augmented the code with a procedure detecting a kernel K, the corresponding live emerging and overflow jobs (l and o, resp.), and the corresponding intersection $\delta(K,l)$. In this way, for every created J-schedule σ, we were able to calculate the objective value ($L_o(\sigma)$) and the lower bounds $L_o(\sigma) - p_l$ and $L_o(\sigma) - \delta(K,l)$ from Lemmas 3 and 4, respectively.

We have created instances for $n = 50, 200, 800, 3200$ ($n = 50, 100, 200, 500, 1000, 2000$, resp.), with the processing times from the interval $[1, 50]$ ($[1, 100]$, resp.). We have created 20 instances for each above n, in total 200 instances. For more than half of these instances, in the created J-schedules no emerging job existed. Hence these instances were solved optimally due to Corollary 2. For the rest of the instances, there existed an emerging job l and kernel K. However, importantly, the corresponding $\delta(K,l)$ was too insignificant, so that most of these instances were solved within a factor of 1.009 of the optimum objective value, whereas the worst approximation ratio was less than 1.03.

A detailed description of the experimental data can be found in Tables 1–10. Each table represents the randomly generated 20 problem instances with a particular $p_{\max}$ and n. The problem instances that were solved optimally (due to the nonexistence of the live emerging job) have no specific entries. For the rest of the problem instances, the tables, besides the approximation ratio of the obtained solution

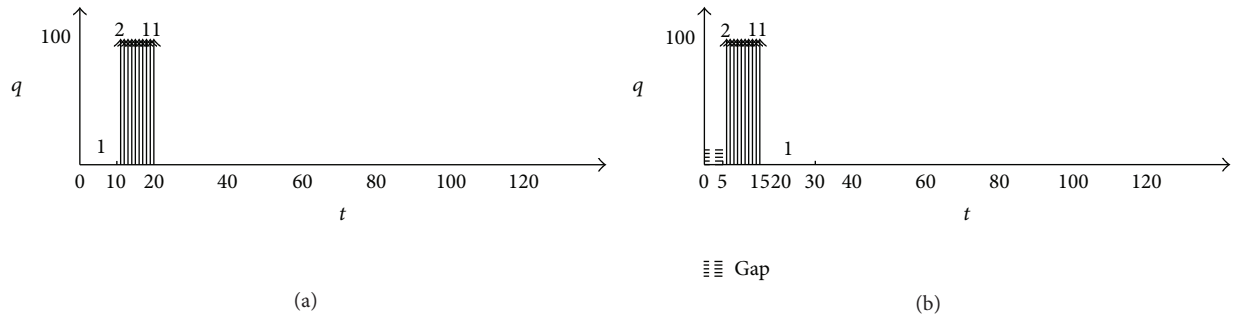

Figure 2: Second instance. On (a) the J-schedule σ and on (b) the optimal schedule S^*.

Table 1: 20 instances of 50 jobs. The processing time p drawn from the interval [1, 50]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
9	17	42	638	88	30	602	100	467	702	59	41	769	728	17.7	1.0563
20	4	28	1236	108	31	1069	31	0	1100	27	4	1101	1096	274	1.0036

Table 2: 20 instances of 200 jobs. The processing time p drawn from the interval [1, 50]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
5	28	38	4267	843	6	3187	818	283	4005	811	7	4288	4281	611.5	1.0016
7	64	42	3520	1200	30	3470	411	0	3881	393	18	4005	3987	221.5	1.0045

Table 3: 20 instances of 800 jobs. The processing time p drawn from the interval [1, 50].

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
All instances were solved optimally															

Table 4: 20 instances of 3200 jobs. The processing time p drawn from the interval [1, 50]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
2	125	50	62627	30505	1	60459	9222	0	69681	9184	38	69681	69643	1832.7	1.0005

Table 5: 20 instances of 50 jobs. The processing time p drawn from the interval [1, 100]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
1	688	89	935	817	82	830	65	1406	895	−13	78	2301	2223	28.5	1.0350
3	94	72	1291	155	34	164	62	2162	226	−9	71	2388	2317	32.6	1.0306
5	705	85	1795	929	31	953	36	1373	989	−24	60	2362	2302	38.3	1.0260
6	1032	82	2463	2291	43	2326	−7	0	2319	−35	28	2319	2291	81.8	1.0122
7	1737	89	2014	1877	36	1882	69	444	1951	−5	74	2395	2331	31.5	1.0274
8	1366	66	1925	1544	88	1573	29	753	1602	−29	58	2355	2297	39.6	1.0252
10	2067	62	2269	2127	52	2146	14	180	2160	−19	33	2340	2307	69.9	1.0143
16	291	79	1454	320	27	355	27	1971	382	−35	62	2353	2291	36.9	1.0270

Table 6: 20 instances of 100 jobs. The processing time p drawn from the interval [1, 100]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
2	1526	98	2515	1590	45	1715	−63	3001	1652	−125	62	4653	4591	74	1.0135
4	1137	30	1967	1189	70	1200	4	3516	1204	−11	15	4720	4705	313.6	1.0031
8	3958	68	4999	4697	10	4716	19	0	4735	−19	38	4735	4697	123.6	1.0080
12	110	78	379	139	33	139	51	4577	190	0	51	4767	4716	92.4	1.0108
13	1169	71	3905	1209	46	1317	−70	3399	1247	−108	38	4646	4608	121.2	1.0082
15	2518	80	4720	4197	55	4218	26	498	4244	−21	47	4742	4695	99.8	1.0100
17	130	76	823	238	24	203	24	4513	227	−25	49	4740	4691	95.1	1.0104
20	1402	42	4239	3227	75	3258	7	1458	3265	−31	38	4723	4685	123.2	1.0081

Table 7: 20 instances of 200 jobs. The processing time p drawn from the interval [1, 100]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
3	7688	83	9677	9389	22	9422	16	0	9438	−33	49	9438	9389	191.6	1.0052
5	4493	80	5774	4621	68	4658	7	4764	4665	−37	44	9429	9385	213.2	1.0046
6	6641	92	7665	6745	87	6812	−30	2610	6782	−67	37	9392	9355	252.8	1.0039
9	3981	61	6226	4114	76	4157	15	5265	4172	−43	58	9437	9379	161.7	1.0061
12	2325	51	8839	7306	33	7319	27	2103	7346	−13	40	9449	9409	235.2	1.0042
13	1304	100	9640	9030	22	9101	2	321	9103	−71	73	9424	9351	128	1.0078
14	3334	74	7872	3374	21	3439	4	5983	3443	−65	69	9426	9357	135.6	1.0073
15	32	97	2879	205	83	207	77	9215	284	−2	79	9499	9420	119.2	1.0083
17	1529	94	9466	8700	53	8713	76	709	8789	−13	89	9498	9409	105.7	1.0094
18	3792	77	8322	5891	95	5896	3	3526	5899	−5	8	9425	9417	1177.1	1.0008
19	3987	57	7815	5509	81	5529	−3	3893	5526	−20	17	9419	9402	553	1.0018
20	3853	78	9696	8813	26	8863	5	559	8868	−55	60	9427	9367	156.1	1.0064

Table 8: 20 instances of 500 jobs. The processing time p drawn from the interval [1, 100]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
1	2737	30	24074	21250	74	21266	5	2065	21271	−16	21	23336	23315	1110.2	1.0009
2	10786	95	18198	11566	87	11656	−4	11675	11652	−90	86	23327	23241	270.2	1.0037
3	2469	100	6761	2531	22	2553	74	20778	2627	−22	96	23405	23309	242.8	1.0041
5	28	11	16601	31	35	32	7	23299	39	−1	8	23338	23330	2916.2	1.0003
6	187	77	9189	413	39	417	35	22914	452	−4	39	23336	23297	597.3	1.0016
8	6251	94	23315	21904	26	21934	62	1397	21996	−30	92	23393	23301	253.2	1.0039
9	6313	72	21427	18378	54	18427	12	4904	18439	−49	61	23343	23282	381.6	1.0026
11	12208	70	23060	19855	67	19866	36	3465	19902	−11	47	23367	23320	496.1	1.0020
16	11401	55	24586	23297	48	23331	9	0	23340	−34	43	23340	23297	541.7	1.0018
20	3181	88	17110	8493	39	8541	−1	14790	8540	−48	47	23330	23283	497.5	1.0020

Table 9: 20 instances of 1000 jobs. The processing time p drawn from the interval [1, 100]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^σ	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
1	48087	82	77454	54385	40	54441	−5	−4441	54436	−56	51	49995	49944	979.29	1.0010
5	39909	22	99943	53552	74	53591	−15	−3591	53576	−39	24	49985	49961	2081.71	1.0005
9	2441	92	3451	2879	66	2882	26	47118	2908	−3	29	50026	49997	1724.03	1.0006

Table 10: 20 instances of 2000 jobs. The processing time p drawn from the interval [1, 100]. The instances that are not shown in the table were solved optimally.

N	Emerging job			Overflow job						LB_1	δ	T^{σ}	LB_2	κ	Aprox. ratio
	r_l	p_l	d_l	r_0	p_0	d_0	L_0	q_0	C_0						
1	48087	82	77454	54385	40	54441	−5	45559	54436	−56	51	99995	99944	1959.69	1.0005
5	39909	22	99943	53552	74	53591	−15	46409	53576	−39	24	99985	99961	4165.04	1.0002
9	34418	98	77808	54170	73	54182	6	45818	54188	−12	18	100006	99988	5554.89	1.0002
13	29865	51	99008	98068	9	98072	28	1928	98100	−4	32	100028	99996	3124.88	1.0003
17	53804	20	71630	54118	16	54165	−30	45835	54135	−47	17	99970	99953	5879.59	1.0002
18	27503	39	67807	30462	7	30474	24	69526	30498	−12	36	100024	99988	2777.44	1.0004
19	17473	86	85224	67046	74	67060	58	32940	67118	−14	72	100058	99986	1388.69	1.0007

due to Theorem 6 (A.R.), specify the parameters of the live emerging job l and the overflow job o for the earliest encountered kernel $K \in \sigma$. In addition, the lateness and the completion time of the overflow job o, the makespan of σ, $\delta(K, l)$, and the corresponding κ are specified. The lower bounds from Lemma 4 in terms of lateness and makespan, respectively, are denoted by LB_1 and LB_2, respectively. As it can be seen from the tables, these lower bounds turned out to be quite strong for the tested problem instances, because of their closeness to the objective value in σ (represented in columns labeled by L_o for the maximum lateness and T^{σ} for the makespan).

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. R. Jackson, "Schedulig a production line to minimize the maximum tardiness," Manegement Scince Research Project, University of California, Los Angeles, Calif, USA, 1955.

[2] L. Schrage, "Obtaining optimal solutions to resource constrained network scheduling problems," unpublished manuscript, 1971.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, Calif, USA, 1979.

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. G. H. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[5] P. Bratley, M. Florian, and P. Robillard, "On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/G/F_{\max})$ problem," *Naval Research Logistics Quarterly*, vol. 20, no. 1, pp. 57–67, 1973.

[6] C. N. Potts, "Analysis of a heuristic for one machine sequencing with release dates and delivery times," *Operations Research*, vol. 28, no. 6, pp. 1436–1441, 1980.

[7] L. A. Hall and D. B. Shmoys, "Jackson's rule for single-machine scheduling: making a good heuristic better," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 22–35, 1992.

[8] G. McMahon and M. Florian, "On scheduling with ready times and due dates to minimize maximum lateness," *Operations Research*, vol. 23, no. 3, pp. 475–482, 1975.

[9] J. Carlier, "The one-machine sequencing problem," *European Journal of Operational Research*, vol. 11, no. 1, pp. 42–47, 1982.

[10] J. Grabowski, E. Nowicki, and S. Zdrzałka, "A block approach for single-machine scheduling with release dates and due dates," *European Journal of Operational Research*, vol. 26, no. 2, pp. 278–285, 1986.

[11] M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan, "Scheduling unit-time tasks with arbitrary release times and deadlines," *SIAM Journal on Computing*, vol. 10, no. 2, pp. 256–269, 1981.

[12] N. Vakhania, "Single-machine scheduling with release times and tails," *Annals of Operations Research*, vol. 129, pp. 253–271, 2004.

[13] N. Vakhania, "Scheduling jobs with release times preemptively on a single machine to minimize the number of late jobs," *Operations Research Letters*, vol. 37, no. 6, pp. 405–410, 2009.

[14] N. Vakhania, "A study of single-machine scheduling problem to maximize throughput," *Journal of Scheduling*, vol. 16, no. 4, pp. 395–403, 2013.

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

[16] B. Simons, "Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines," *SIAM Journal on Computing*, vol. 12, no. 2, pp. 294–299, 1983.

[17] B. B. Simons and M. K. Warmuth, "A fast algorithm for multiprocessor scheduling of unit-length jobs," *SIAM Journal on Computing*, vol. 18, no. 4, pp. 690–710, 1989.

[18] N. Vakhania, "A better algorithm for sequencing with release and delivery times on identical machines," *Journal of Algorithms*, vol. 48, no. 2, pp. 273–293, 2003.

[19] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Management Science*, vol. 35, no. 2, pp. 164–176, 1989.

[20] W. Brinkkotter and P. Brucker, "Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments," *Journal of Scheduling*, vol. 4, no. 1, pp. 53–64, 2001.

[21] A. Gharbi and M. Labidi, "Jackson's semi-preemptive scheduling on a single machine," *Computers & Operations Research*, vol. 37, no. 12, pp. 2082–2088, 2010.

[22] F. D. Croce and V. T'kindt, "Improving the preemptive bound for the single machine dynamic maximum lateness problem," *Operations Research Letters*, vol. 38, no. 6, pp. 589–591, 2010.

[23] J. Carlier and E. Pinson, "Jakson's pseudo preemptive schedule for the Pm/ri, qi/C_{max} problem," *Annals of Operations Research*, vol. 83, pp. 41–58, 1998.

[24] N. Vakhania and E. Shchepin, "Concurrent operations can be parallelized in scheduling multiprocessor job shop," *Journal of Scheduling*, vol. 5, no. 3, pp. 227–245, 2002.

[25] M. Perregaard and J. Clausen, "Parallel branch-and-bound methods for the job-shop scheduling problem," *Annals of Operations Research*, vol. 83, pp. 137–160, 1998.

[26] A. Condotta, S. Knust, and N. V. Shakhlevich, "Parallel batch scheduling of equal-length jobs with release and due dates," *Journal of Scheduling*, vol. 13, no. 5, pp. 463–477, 2010.

[27] N. Vakhania and F. Werner, "Minimizing maximum lateness of jobs with naturally bounded job data on a single machine in polynomial time," *Theoretical Computer Science*, vol. 501, pp. 72–81, 2013.