Chapter 4

Efficient Heuristics for Scheduling with Release and Delivery Times

Nodari Vakhania

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.69223

Abstract

In this chapter, we describe efficient heuristics for scheduling jobs with release and delivery times with the objective to minimize the maximum job completion time. These heuristics are essentially based on a commonly used scheduling theory in Jackson's extended heuristic. We present basic structural properties of the solutions delivered by Jackson's heuristic and then illustrate how one can exploit them to build efficient heuristics.

Keywords: combinatorial optimization, heuristic algorithm, scheduling theory, time complexity, approximation algorithm

1. Introduction

The *combinatorial optimization* problems have emerged in late 40s of last century due to a rapid growth of the industry and new arisen demands in efficient solution methods. Modeled in mathematical language, a combinatorial optimization problem has a finite set of the so-called *feasible solutions*; this set is determined by a set of restrictions that naturally arise in practice. Usually, there is an objective function in which domain is the latter set. One aims to determine a feasible solution that gives an extremal (minimal or maximal) value to the objective function, the so-called *optimal solution*. Since the number of feasible solutions is typically finite, 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. The main issue here is that a brutal enumeration of all feasible solutions might be impossible in practice.

There are two distinct classes of combinatorial optimization problems, class *P* of polynomially solvable ones and *NP*-hard problems. For a problem from class *P*, there exists an efficient

(polynomial in the *size* of the problem) algorithm. But no such algorithm exists for an *NP*-hard problem. The number of feasible solutions of an *NP*-hard optimization problem grows exponentially with the size of the input (which, informally, is the amount of the computer memory necessary to represent the problem data/parameters). Furthermore, all *NP*-hard problems have a similar *computational* (*time*) *complexity*, in the sense that if there will be found an efficient polynomial-time algorithm for any of them, such an algorithm would yield another polynomial-time algorithm for any other *NP*-hard problem. On the positive side, all *NP*-hard problems belong to the *class NP* that guarantees that any feasible schedule to an *NP*-hard problem can be found in polynomial time. It is believed that it is very unlikely that an *NP*-hard problem can be solved in polynomial time (whereas an exact polynomial-time algorithm with a reasonable real-time behavior exists for a problem in class *P*). Hence, it is natural to think of an approximation solution method.

Thus, approximation algorithms are most frequently used for the solution of *NP*-hard problems. Any *NP*-hard problem has a nice characteristic, that is, any feasible solution can be created and verified in polynomial time, like for problems in class *P*. A *greedy* algorithm creates in this way a single feasible solution by iteratively taking the next "most promising" decision, until a complete solution is created. These decisions are normally taken in a low-degree polynomial/constant time. Since the total number of iterations equals to the number of objects in a given problem instance, the overall time complexity of a greedy algorithm is low. Likewise, *heuristic* algorithms reduce the search space creating one or more feasible solutions in polynomial time. Greedy and heuristic algorithms are simplest approximation algorithms. It is easy to construct such an algorithm for both polynomial and *NP*-hard problems. It may deliver an optimal solution to a problem from calls *P*, but it is highly unlikely that a heuristic optimal algorithm may exist for an *NP*-hard problem. A greedy algorithm reduces the potential search space by taking a unique decision for the extension of the current partial solution in each of the *n* iterations. A simplest heuristic algorithm is greedy, though there are more sophisticated heuristic algorithms that use different search strategies. In general, an approximation algorithm may guarantee some worst-case behavior measured by its *performance ratio*: the ratio of the value of objective function of the worst solution that may deliver the algorithm to the optimal objective value (a real number greater than 1).

Scheduling problems are important combinatorial optimization problems. A given set of requests called *jobs* are to be performed (scheduled) on a finite set of resources called *machines* (or *processors*). The objective is to determine the processing order of jobs on machines in order to minimize or maximize a given objective function. Scheduling problems have a wide range of applications from production process to computer systems optimization.

Simple greedy heuristics that use some priority dispatching rules for the for taking the decisions can be easily constructed and adopted for scheduling problems. An obvious advantage of such heuristics is their rapidness, and an obvious disadvantage is a poor solution quality. The generation of a better solution needs more computational and algorithmic efforts. A *Global search* in the feasible solution space guarantees an optimal solution, but it can take inadmissible computational time. A *local* (*neighborhood*) *search* takes reasonable computational time, and the solution which it gives is locally best (i.e., best among all considered neighbor solutions).

Simulated annealing, tabu-search, genetic algorithms, and beam search are examples of local search algorithms (for example, [16, 22, 23, 25]). These algorithms reduce the search space, and at the same time, their search is less restricted than that of simple heuristic (dispatching) algorithms, giving, in general, better quality solutions than simple greedy algorithms. Global search methods include (exact) implicit enumerative algorithms and also approximative algorithm with embedded heuristic rules and strategies (for example, [1, 20, 29, 38, 4]). Normally, global search algorithms provide the solutions with the better quality than the local search algorithms, but they also take more computer time.

This chapter deals with one of the most widely used greedy heuristics in scheduling theory. The generic heuristic for scheduling jobs with release and delivery times on a single machine to minimize the maximum job completion time is named after Jackson [21] (the heuristic was originally proposed for the version without release times, and then it was extended for the problem with release times by Schrage [30]). Jackson's heuristic (J-heuristic, for short), iteratively, at each scheduling time t (given by job release or completion time), among the jobs released by time t schedules one with the largest delivery time. This 2-approximation heuristic is important on its own right, and it also provides a firm basis for more complex heuristic algorithms that solve optimally various scheduling problems that cannot be solved by a greedy method.

In this chapter, we give a brief overview of heuristic algorithms that are essentially based on J-heuristic. Then, we go into the analysis of the schedules created by J-heuristic (J-schedule), showing their beneficial structural properties. They are helpful for a closer study of the related problems, which, in turn, may lead to better solution methods. We illustrate how the deduced structural properties can be beneficially used by building an adaptive heuristic algorithm for our generic scheduling problem with a more flexible worst-case performance ratio than that of J-heuristic.

The next section consists of four subsections. In Section 2, we first describe our basic scheduling problem, Jackson's heuristic, other related heuristics, and real-life applications of the scheduling problem. In Section 3, we study the basic structural properties of the schedules constructed by Jackson's heuristic. In Section 4, we derive a flexible worst-case performance estimation for the heuristic, and in Section 5, we construct a new adaptive heuristic based on Jackson's heuristic. Section 6 concludes the chapter with final remarks.

2. Preliminaries

2.1. Problem formulation

Our generic single-machine scheduling problem can be described as follows. We have n jobs from set J and a single machine. Job $j \in J$ is characterized by its *release time* r_j, a time moment when it becomes available. Once a released job is assigned to the machine, it takes p_j time units of uninterrupted processing time on that machine. Here, we have a basic restriction that the machine can process no more than one job at any time moment. Once job j completes its

processing on the machine, it 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). We wish to minimize the maximum job full completion time.

The problem is known to be strongly *NP* hard (Garey and Johnson [13]). According to the conventional three-field notation introduced by Graham et al. [18], 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 the third field specifies objective criteria.

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).

We may note that, besides job lateness, there are other due-date-oriented objective criteria. A common one is *the number of late* jobs, where job is late if it completes behind its due date. Here, the number of late jobs is to be minimized. In the *feasibility* version of the problem, one looks for a schedule with no late job. Obviously, if in an optimal solution of the minimization version, the maximum job lateness is no more than 0, then it is a feasible solution of the feasibility version as well; otherwise (the maximum job lateness is positive), there exists no feasible solution to the feasibility version. Vice versa, an algorithm for the feasibility problem can be used to solve the minimization version: we iteratively increase due dates of all jobs until we find a feasible schedule with the modified due dates. Note that the min-max job lateness obtained in this way depends on the maximum job processing time $p_{\max}$ and n so that we will need to apply a feasibility algorithm $O(np_{\max})$ times. But by using binary search, the cost can be reduced to $O(log(np_{\max}))$.

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 $1|r_j|L_{\max}$, we may create an equivalent instance of $1|r_j,q_j|C_{\max}$ by introducing job delivery times, $q_j = D - d_j$, taking a suitably large constant D (any number larger than the maximum job due date would work). It can be easily seen by the equivalence of these instances (if 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. [2] for more details). Because of the equivalence, both above formulations might be used interchangeably.

2.2. Description of J-heuristic

Now, we describe Jackson's greedy heuristic (J-heuristic) in detail that works on n scheduling times (at every scheduling time, the next job is scheduled on the machine). Initially, the earliest scheduling time is set to the minimum job release time. Iteratively, among all jobs released by a given scheduling time, a job with the maximum delivery time is scheduled on the machine (ties might be broken by selecting any longest available job). Once a job completes on the

machine, the next scheduling time is set to the maximum between the completion time of that job and the minimum release time of a yet unscheduled job.

Since the heuristic always schedules an earliest released job every time, the machine becomes idle and it creates no gap that can be avoided. The time complexity of the heuristic is $O(n \log n)$ as at every n scheduling times, the search for a maximal element in an ordered list is carried out.

The heuristic is easily expendable for multiprocessor and preemptive scheduling problems with release and delivery (due) times. For m identical parallel processor case, a ready job with the largest tail (or smallest due date) is repeatedly determined and is scheduled on the processor with the minimal completion time ties being broken by selecting the processor with the minimal index. For the sake of conciseness, we below refer to that processor as the *active* one.

Multiprocessor J-heuristic
$U := J$; $t := \min\{r_j | j \in U\}$
while $U \neq \emptyset$ **do**
begin
find job $j^* \in \{j \in U | r_j \leq t\}$ with the largest delivery time q_{j^*} and schedule it at time t on the corresponding active processor; $U := U \setminus \{j^*\}$;

update the current active processor and set t to the maximum between the completion time of that processor and minimal job release time in set U
end

We illustrate a 3-processor J-schedule in **Figure 1** for eight jobs with the parameters as specified in the table as follows:

$r_1 = 0$	$p_1 = 4$	$q_1 = 30$
$r_2 = 0$	$p_2 = 5$	$q_2 = 25$
$r_3 = 5$	$p_3 = 3$	$q_3 = 20$
$r_4 = 8$	$p_4 = 6$	$q_4 = 15$
$r_5 = 8$	$p_5 = 10$	$q_5 = 22$
$r_6 = 15$	$p_6 = 2$	$q_6 = 5$
$r_7 = 20$	$p_7 = 4$	$q_7 = 30$
$r_8 = 25$	$p_8 = 7$	$q_8 = 10$

As it can be seen in **Figure 1**, J-heuristic creates idle time intervals (the gaps) on all three processors constructing an optimal schedule with makespan 54. Note that job 7 realizes the maximum objective value 54 being scheduled on processor 2 (we call such job the overflow job

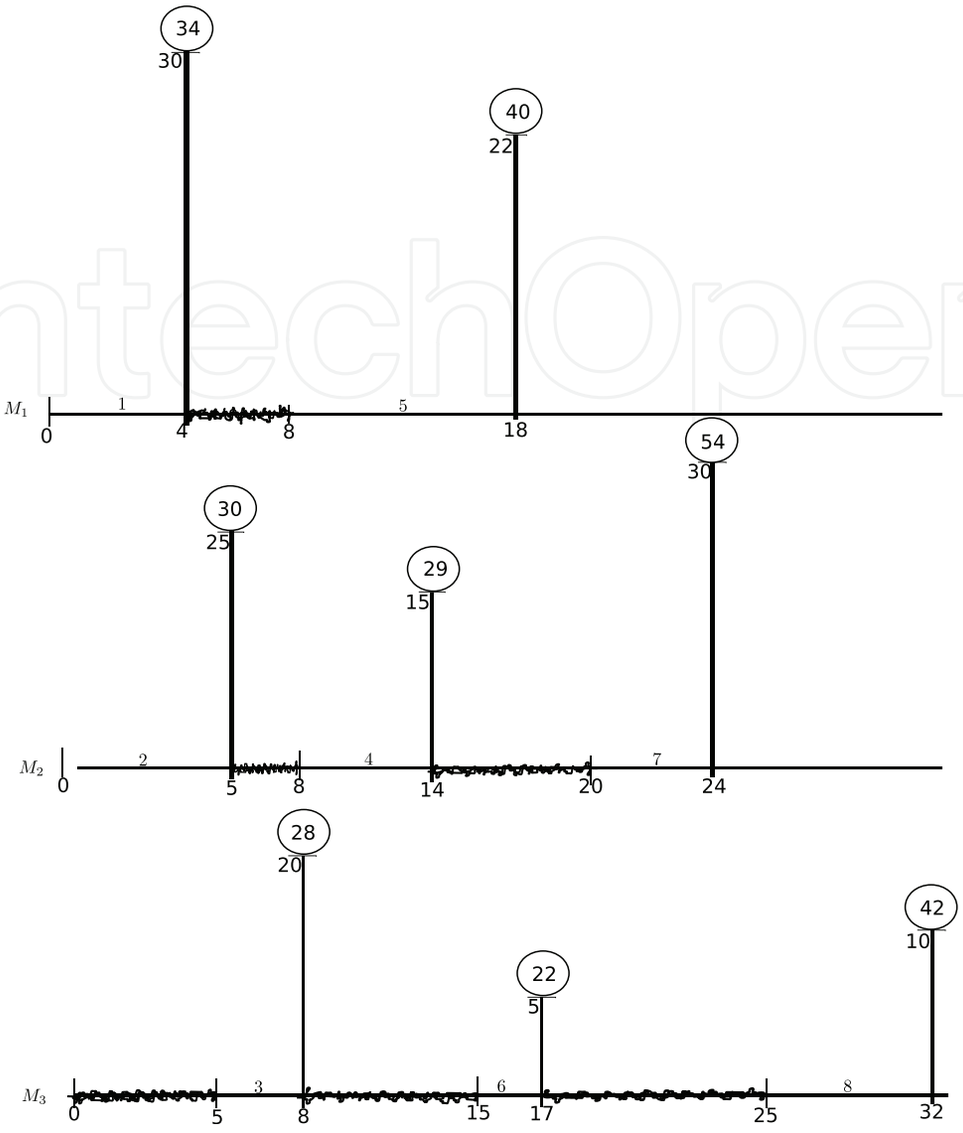

Figure 1. A 3-processor J-schedule. Zig-zag lines represent gaps, and the numbers within the circles are job full completion times.

as we define a bit later), the completion time of processor 2 is 24 (that of job 7), whereas the full completion time of job 7 is 54.

For preemptive version of J-heuristic, every currently executed job is interrupted at the earliest time moment when a more urgent jobs get released. It is well-known and also easily seen that the preemptive version of J-heuristic delivers an optimal solution for the corresponding preemptive scheduling problem.

2.3. Overview of related heuristics

As mentioned earlier, J-heuristic turned out to be highly flexible in the sense that its different modifications have been used for the solution of different scheduling problems. Potts [27] has proposed an extension of the heuristic. His algorithm repeatedly applies J-heuristic $O(n)$ times

and obtains an improved approximation ratio of 3/2 at the cost of an increase by a factor of $O(n)$ time complexity. Hall and Shmoys [19], also based on J-heuristic, have developed another 4/3-approximation polynomial-time algorithm with the same time complexity of $O(n^2 \log n)$ for the version of our problem $1|r_j,q_j|C_{\max}$ with precedence relations. Garey et al. [14] have modified the heuristic as another more sophisticated $O(n \log n)$ heuristic 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 complete by their deadlines is looked for). This result was extended to the version of problem $1|r_j,q_j|C_{\max}$ with two possible job processing times in an $O(n^2 \log n)$ algorithm described in [34]. For another relevant criterion, 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 was proposed in [35]. Without preemptions, an $O(n^2 \log n)$ algorithm for the case when all jobs have equal length was proposed in [37].

Multiprocessor version of J-heuristic has been used as a basis for the solution of 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 [31] and Simons and Warmuth [32], respectively. Using the J-heuristic as a schedule generator, an $O(q_{\max} mn \ \log n + O(mvn))$ algorithm for the minimization version of the latter problem was proposed in [33], where $q_{\max}$ is the maximum job delivery time and $v < n$ is a parameter. With the objective to minimize the number of late jobs on a group of identical processors, an $O(n^3 \log n \log p_{\max})$ non-preemptive algorithm for equal-length jobs was proposed in [36].

J-heuristic can be efficiently used for the solution of shop scheduling problems. Using J-heuristic as a schedule generator, McMahon and Florian [24] and Carlier [5] have proposed efficient enumerative algorithms for $1|r_j,q_j|C_{\max}$. Grabowski et al. [17] use the heuristic for the obtainment of an initial solution in another enumerative algorithm for the same problem.

The problem $1|r_j,q_j|C_{\max}$ naturally arises in job-shop scheduling problems as an auxiliary problem for the derivation of strong lower bounds. By ignoring the potential yet unresolved conflicts on all the machines except a selected machine M, the corresponding disjunctive graph defines an auxiliary instance of problem $1|r_j,q_j|C_{\max}$ on machine M, where every task o to be performed on that machine is characterized by an early starting time (defined by the early completion times of its predecessor-tasks) that is set as its release time r_o and the tail or the delivery time q_o (determined by the processing times of the predecessor-tasks of task o). In multiprocessor job-shop scheduling problems, a single machine is replaced by a group of parallel machines, and the corresponding multiprocessor version of problem $1|r_j,q_j|C_{\max}$ is derived. For the purpose of a lower bound, preemptive version of the above problems with release and delivery times might be considered and preemptive J-heuristic can then be applied. For relevant studies on a classical job-shop scheduling problem, see, for example, Carlier [5], Carlier and Pinson [6], Brinkkotter and Brucker [3], and more recent works of Gharbi and Labidi [15] and Della Croce and T'kindt [12] and for multiprocessor job-shop scheduling problem with identical machines, see Carlier and Pinson [7]. This approach can also be extended for the case when parallel machines are unrelated [38].

J-heuristic can also be useful for parallelizing the computations in scheduling job shop [26] and also for the parallel batch scheduling problems with release times [10].

2.4. Some other applications

Besides the above-mentioned applications in multiprocessor, shop, and batch scheduling problems, our problem has numerous immediate real-life applications in various production chains, CPU time sharing in operating systems (jobs being the processes to be executed by the processor), wireless sensor network transmission range distribution (where jobs are mobile devices with their corresponding ranges that can be modeled as release and due dates), and important transportation problems such as traveling salesman's and vehicle routing problems with time windows. The reader may wish to have a look at reference [28] for an extensive overview of the variations of the vehicle routing problem, and we also refer to [11, 41] for more recent related work.

Our scheduling problem can be used for the solution of the latter transportation problems. Let us first describe these problems briefly. There are, say, n customers or cities and one special location called depot. The distances between any pair of locations are known. The goods are to be distributed from depot to the customers using one or more (identical) vehicles. There are certain restrictions on how this distribution should be done that define set of all feasible solutions to the problem. A general notion is a tour carried out by a vehicle that initiates at depot, visits some of the customers, and returns to depot. All customers must be served, i.e., every customer is to be included in exactly one tour. There may be additional restrictions such as vehicle capacity constraints and customer requests. And another basic constraint, which relates these transportation problems with our scheduling problem, is that every customer can be served only within a certain time interval, whereas there is also a valid time interval given for the depot.

A common objective is to minimize the total service/travel time of all the vehicles. Whereas in the basic setting, it is straightforward to find a feasible solution, with time windows, this task is not obvious, in fact, there may exist no feasible solution. If it exists, then one aims to minimize the number of used vehicles and then construct the corresponding number of tours with the objective to minimize total service time.

Associating with every customer and the depot a unique job and with the corresponding time window the release and due dates of that job, we arrive at a corresponding scheduling problem, an instance of $1|r_j|L_{\max}$. Let us consider the feasibility version of this problem (in which a solution with no positive lateness is looked for). Note that if there is no feasible solution to that feasibility version, then there exists no feasible solution to the corresponding vehicle routing problem with a single vehicle. Then, we may consider the feasibility problem with two identical machines $P2|r_j|L_{\max}$ and so on, with k identical machines $Pk|r_j|L_{\max}$, until a feasible solution is found. We may use a binary search within the interval $[1, m]$ instead when an upper limit m on the maximum number of vehicles is known (otherwise, we set m to a sufficiently large magnitude). In case, there exists a feasible solution for $k = m$, once the minimum k is found, the corresponding k tours minimizing the total travel time might be constructed.

3. The structure of J-schedules

Previous section's brief survey clearly indicates importance of our scheduling problem and the power and flexibility of J-heuristic as well. Whenever the direct application of J-heuristic for the solution of the problem is concerned and the solution quality is important, the worst-case bound of two may not be acceptable. Besides, J-heuristic may not solve the feasibility version of our problem even though there may exist a feasible solution with no positive maximum lateness. To this end, there are two relevant points that deserve mentioning. On the one hand, the practical behavior of the heuristic might be essentially better than this worst-case estimation [40]. On the other hand, by carrying out structural analysis of J-schedules, it is possible to obtain a better, more flexible worst-case bound, as we show in the next section. In this section, we introduce some basic concepts that will help us in this analysis.

Let us denote by σ, the schedule obtained by the application of J-heuristic to the originally given problem instance (as we will see later, this heuristic can also be beneficially applied to some other derived problem instances). Schedule σ, and, in general, any J-schedule, may contain a *gap*, which is its maximal consecutive time interval in which the machine is idle. We shall 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, respectively) denotes the starting (completion, respectively) time of job j.

Let us call a *block*, a maximal consecutive part of a J-schedule, is consisting of the successively scheduled jobs without any gap in between (preceded and succeeded by a gap).

Now, we give some necessary concepts from [33] that will help us to expose useful structural properties of the J-schedules.

Given a J-schedule S, let i be a job that realizes the maximum job lateness in S, i.e., $L_i(S) = \max_j\{L_j(S)\}$. 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 to the number of the overflow jobs in it. Furthermore, since any kernel belongs to a single block, it may contain no gap.

If schedule σ is not optimal, there must exist a job less urgent than o, scheduled before all jobs of kernel K that delays jobs in K (see Lemma 1 a bit later). 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 (see Lemma 1 below), 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.

Now, we can give some optimality conditions for J-schedules. The proofs of Lemmas 1 and 2 can be found in references [33, 39], respectively. Lemma 4 is obtained as a consequence of Lemmas 1 and 2, though the worst-case bound of two was also known earlier.

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, it is optimal.*

From Lemma 1, we obtain the following corollary:

Corollary 1. *If schedule σ contains a kernel with no live emerging job* $(E(\sigma) = \varnothing)$*, then σ is optimal.*

Observe that the conditions of the Lemma 1 and Corollary 1 are satisfied for our first problem instance of **Figure 1**. We also illustrate the above-introduced definitions on a (1-processor) problem instance of $1|r_j,q_j|C_{\max}$ with 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, have the equal processing time 1, and the delivery time 100. These data completely define our problem instance.

The initial J-schedule σ consists of a single block, in which 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. 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 as its kernel K starts at time 100 which is more than $r(K) = 10$. Furthermore, the condition of Corollary 1 is also not satisfied for schedule σ, and it is not optimal. The optimal schedule S^* has the makespan 120, in which the live emerging job 1 is rescheduled behind all kernel jobs.

From here on, we use T^S for the makespan (maximum full job completion time) of J-schedule S and T^* (L^*_{max}, respectively) for the optimum makespan (lateness, respectively).

Lemma 2. $T^\sigma - T^* < p_l$ $\left(L^\sigma_{max} - L^*_{max} < p_l\right)$*, where l is the live emerging job for kernel* $K \in \sigma$.

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

Note that Lemma 2 implicitly defines a lower bound of $T^\sigma - p_l$ derived from the solution of the non-preemptive J-heuristic, which 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, respectively) stands again for the live emerging (overflow, respectively) job for kernel K). Then, the next lemma follows from the observation that $\delta(K,l)$ is another (more accurate than p_l) estimation for the delay of the earliest scheduled job of kernel K.

Lemma 3 *$L^* = T^\sigma - \delta(K,l)$ ($L_o(\sigma) - \delta(K,l)$, respectively) is a lower bound on the optimal job makespan T^* (lateness $L^*_{\max}$, respectively).*

Lemma 4 *J-heuristic gives a 2-approximate solution for $1|r_j,q_j|C_{\max}$, i.e., $T^\sigma/T^* < 2$.*

Proof. If there exists no live emerging job l for $K \in \sigma$, then σ is optimal by Corollary 1. Suppose job l exists; clearly, $p_l < T^*$ (as l has to be scheduled in S^* and there is at least one more (kernel) job in it). Then, by Lemma 2,

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

□

4. Refining J-heuristic's worst-case estimation

From Lemma 2 of the previous section, we may see that the quality of the solution delivered by J-heuristic is somehow related with the maximum job processing time $p_{\max}$ in a given problem instance. If such a long job turns out to be the live emerging job, then the corresponding forced delay for the successively scheduled kernel jobs clearly affects the solution quality. We may express the magnitude $p_{\max}$ as a fraction the optimal objective value and derive a more accurate approximation ratio. It might be possible to deduce this kind of relationship priory with a good accuracy. Take, for instance, a large-scale production where the processing time of an individual job is small enough compared to an estimated total production time T.

If this kind of prediction is not possible, we can use the bound from Lemma 3 by a single application of J-heuristic and represent $p_{\max}$ as its fraction κ (instead of representing it as a fraction of an unknown optimal objective value). Then, we can give an explicit expression of the heuristic's approximation ratio in terms of that fraction. As we will see, J-heuristic will always deliver a solution within a factor of $1 + 1/\kappa$ of the optimum objective value. Alternatively, we may use a lower bound on the optimal objective value L^* from Lemma 3 (as T^* may not be known). Let $\kappa > 1$ be such that $p_l \le T^*/\kappa$, i.e., $\kappa \le T^*/p_l$. Since L^* is a lower bound on T^* ($L^* \le T^*$), we let $\kappa = L^*/p_l$, and thus we have that $\kappa \le T^*/p_l$, i.e., $\kappa = L^*/p_l$ is a valid assignment. Then, note that for any problem instance, κ can be obtained in time $O(n \log n)$.

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

Proof. By Lemma 2,

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

□

In the previous section's example, we had a very long live emerging job that has essentially contributed in the makespan of schedule σ. The resultant J-schedule gave an almost worst-possible performance ratio from Lemma 4 due to a significant intersection $\delta(K,l)$ (close to the magnitude p_l). We now illustrate an advantage of the estimation from Theorem 1. Consider a

slightly modified instance in which the emerging job l remains moderately long (a more typical average scenario). A long emerging job 1 has processing time $p_1 = 10$, release time $r_1 = 0$, and the delivery time $q_1 = 0$. 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 has the makespan 120. The lower bound on the optimum makespan defined by Lemma 2 is hence $120 - 10 = 110$.

The approximation provided by J-heuristic for this problem instance can be obtained from Theorem 1. Based on Theorem 1, 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 $1 + 1/11.5$. Observe that for our second (average) problem instance, J-heuristic gave an almost optimal solution.

5. An adaptive 3/2-approximation heuristic

In Section 2, we have mentioned two $O(n^2 \log n)$ heuristic algorithms from references [19, 27] solving for our generic problem with the approximation ratios 3/2 and 4/3, respectively. In the previous section, we have described a more flexible approximation ratio that was obtained by a single application of J-heuristic.

In this section, we propose an $O(n \log n)$ heuristic that gives approximation ratio 3/2 for a large class of problem instances, which we will specify a bit later. This algorithm, unlike the above-mentioned algorithms, carries out a constant number of calls to J-heuristic (yielding thus the same time complexity as J-heuristic). Recall that the initial J-schedule σ is obtained by a call of J-heuristic for the originally given problem instance. By slightly modifying this instance, we may create alternative J-schedules with some desired property. With the intention to improve the initial J-schedule σ, and more generally, any J-schedule S, jobs in kernel $K = K(S)$ can be restarted earlier.

To this end, we *activate* an emerging job e for kernel K, *that is*, we force job e and all jobs scheduled after kernel K to be scheduled behind K (all these jobs are said to be in the state of activation for K). Technically, we achieve this by increasing the release times of all these jobs to a sufficiently large magnitude, say, $r(K) = \max_{j \in K}\{r_j\}$, so that when J-heuristic is newly applied, neither job e nor any job scheduled after K in S will surpass any job in K, and hence the earliest job in kernel K will start at time $r(K)$.

We call the resultant J-schedule a *complementary* to S schedule and denote it by S_l. Thus, to create schedule S_l, we just increase r_l to $r(K)$ and apply the heuristic again to the modified instance.

Our $O(n \log n)$ heuristic first creates schedule σ, determines kernel $K = K(\sigma)$, and verifies if there exists the live emerging job l; if there is no l, then σ is optimal (Corollary 1). Otherwise, it creates one or more complementary schedules. The first of these complementary schedules is σ_l. If job l remains to be an emerging job in schedule σ_l, then the second complementary schedule $(\sigma_l)_l$, obtained from the first one by activating job l for kernel $K(\sigma_l)$, is created. This operation is repeatedly applied as long as the newly arisen overflow job, that is, the overflow job in the latest created complementary schedule is released within the execution interval of

job l in schedule σ (job l is activated for the kernel of that complementary schedule). The algorithm halts when either l is not an emerging job in the newly created complementary schedule or the overflow job in that schedule is released behind the execution interval of job l in schedule σ. Then, the heuristic determines the best objective value among the constructed J-schedules and halts.

Theorem 2 *The modified heuristic has the performance ratio less than* 3/2.

Proof. In an optimal schedule S^*, either (1) job l remains to be scheduled before the overflow job o of schedule σ (and hence before all jobs of kernel $K = K(\sigma)$) or (2) l is scheduled after job o (and hence after kernel K).

Let E be the set of emerging jobs in schedule σ not including the live emerging job l. In case (1), either σ is already optimal or otherwise $E \neq \emptyset$, and some job(s) from set E are scheduled after kernel K in an optimal schedule S^* (so that job l and the jobs in K are rescheduled, respectively, earlier). Let $P = P(E)$ be the total processing time of jobs in E. Since job l stays before kernel K, $T^\sigma - T^* < P$ (this can be seen similarly as Lemma 2). Let (real) α be such that $P = \alpha p_l$. Since schedule S^* contains jobs of set E and job l, $T^* \geq \alpha p_l + p_l = (1+\alpha)p_l$. We have

$$T^\sigma/T^* < (T^* + \alpha p_l)/T^* = 1 + \alpha p_l/T^* \leq 1 + \alpha p_l/((1+\alpha)p_l) = 1 + \alpha/(1+\alpha). \quad (3)$$

Hence, if $\alpha \leq 1$ (i.e., $P \leq p_l$), then $T^\sigma/T^* < 3/2$.

Suppose now $P > p_l$. Then, $T^* > 2p_l$ and using again Lemma 2

$$T^\sigma/T^* < (T^* + p_l)/T^* < 1 + p_l/(2p_l) = 3/2. \quad (4)$$

It remains to be considered in case (2) when job l is scheduled after (all jobs from) kernel K in schedule S^*. We claim that schedule S^* is again "long enough," i.e., $T^* > 2p_l$. Indeed, consider the J-schedule σ_l. If σ_l is not optimal, then there exists an emerging job in S_l. Similarly as above, in schedule S^*, either (2.1) job l remains before $K(\sigma_l)$ or (2.2) l is scheduled after $K(\sigma_l)$.

In case (2.2), l must be an emerging job in σ_l. If the overflow job in kernel $K(\sigma_l)$ is released after time moment p_l, then $T^* > 2p_l$ as job l is scheduled after the jobs in $K(\sigma_l)$ in schedule $(\sigma_l)_l$. Otherwise, suppose the overflow job in schedule σ_l is released within time interval $(0,p_l)$ (note that it cannot be released at time 0 as otherwise would have originally been included ahead job l by J-heuristic). Without loss of generality and for the purpose of this proof, assume l is an emerging job in schedule σ_l, as otherwise the latter schedule already gives a desired approximation, similarly as in case (1). Because of the same reason, either schedule $(\sigma_l)_l$ gives a desired approximation or otherwise job l remains to be an emerging job (now, $(\sigma_l)_l$), and the heuristic creates the next complementary schedule $((\sigma_l)_l)_l$. We repeatedly apply the same reasoning to the following created complementary schedules as long as the overflow job in the latest created such schedule is released within time interval $(0,p_l)$. Once the latter condition is not satisfied, job l will be started at time moment, larger than p_l in the corresponding complementary schedule. Hence, its length will be at least $2p_l$. Moreover, an optimal schedule S^* must be at least as long as $2p_l$ unless one of the earlier created complementary schedules is optimal. Hence, one of the generated complementary schedules gives a desired approximation.

In case (2.1), if there is no emerging job in $\sigma_{l,}$ then we are done. Otherwise, let E' be the set of emerging jobs in σ_l not including job l. Then similarly as for case (1), there are two sub-cases $P(E') \leq p_l$ and $P(E') > p_l$, in each of which a desired approximation is reached. The theorem is proved.

As to the time complexity of the modified heuristic, note that, in the worst-case, the overflow job in every created complementary schedule is released no later than at time p_l. Then, the algorithm may create up to $n-1$ complementary schedules, and its time complexity will be the same as that of the earlier-mentioned algorithms. However, it is clear that very unlikely, in an instance of $1|r_j,q_j|C_{\max,}$ an "unlimited" amount of jobs are released before time p_l (that would be a highly restricted instance). In average, however, we normally would expect a constant number of such jobs, which, more restrictively, must be overflow jobs in the created complementary schedules (not belonging to kernel $K(\sigma)$). In this case, our heuristic will obviously run in time $O(n\log n)$. We have proved the following result:

Theorem 3. *The modified heuristic runs in time $O(n\log n)$ for any problem instance of $1|r_j,q_j|C_{\max}$ in which the total number of the arisen overflow jobs in all the created complementary schedules released before time p_l is no more than a constant κ (more brutally, for any instance in which the total number of jobs released before time p_l is bounded by κ).*

6. Conclusion

We have described efficient heuristic methods for the solution of a strongly *NP*-hard scheduling problem that, as we have discussed, has a number of important real-life applications. We have argued that it is beneficial as an analysis of the basic structural properties of the schedules created by J-heuristic for the construction of efficient heuristic methods with guaranteed worst-case performance ratios. As we have seen, not only J-heuristic constructs 2-optimal solutions in a low-degree polynomial time, but it is also flexible enough to be served as a basis for other more efficient heuristics. The useful properties of J-schedules were employed in our flexible worst-case performance bound of Section 4 and in the proposed, in Section 5, heuristic algorithm with an improved performance ratio. The latter heuristic is adaptive in the sense that it takes an advantage of the structure of an average problem instance and runs faster for such instances.

We believe that J-schedules possess further useful yet undiscovered properties that may lead to the disclosure of yet unknown insights of the structure of the related problems with release and delivery times. This kind of study was reported in recently published proceedings [8, 9] for the case of a single processor and two allowable job release and delivery times. It still remains open whether basic properties described in these works can be generalized for a constant number of job release and delivery times and for the multiprocessor case. At the same time, some other yet not studied properties even for a single processor and two allowable job release and delivery times may exist. The importance of such a study is emphasized by the fact that the basic single-machine scheduling problem is strongly *NP*-hard and that the version with only two allowable job release and delivery times remains *NP* hard [8].

Author details

Nodari Vakhania

Address all correspondence to: nodari@uaem.mx

Center of Research and Science, UAEMor, Mexico

References

[1] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. Management Science. 1988;**34**:391–401

[2] Bratley P, Florian M, Robillard P. On sequencing with earliest start times and due-dates with application to computing bounds for ($n/m/G/F_{max}$) problem. Naval Research Logistics Quarterly. 1973;**20**:57–67

[3] Brinkkotter W, Brucker P. Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments. Journal of Scheduling. 2001;**4:**53–64

[4] Carballo L, Vakhania N, Werner F. Reducing efficiently the search tree for multiprocessor job-shop scheduling problems. International Journal of Production Research 2013;**51**(23–24)**:**7105–7119. DOI: 10.1080/00207543.2013.837226

[5] Carlier J. The one-machine sequencing problem. European Journal of Operations Research. 1982;**11**:42–47

[6] Carlier J, Pinson E. An algorithm for solving job shop problem. Management Science. 1989;**35**:164–176

[7] Carlier J, Pinson E. Jackson's pseudo preemptive schedule for the $Pm/r_i,q_i/C_{max}$ problem. Annals of Operations Research 1998;**83**:41–58

[8] Chinos E, Vakhania N. Polynomially solvable and NP-hard special cases for scheduling with heads and tails. In: Recent Advances in Mathematics and Computational Science. (MCSS 16). Barcelona, Spain 2016. pp. 141–145. Available from: http://www.wseas.us/e-library/conferences/2016/barcelona/MCSS/MCSS-17.pdf

[9] Chinos E, Vakhania N. Scheduling jobs with two release times and tails on a single machine. International Journal of Mathematical Models and Methods in Applied Sciences 2016;**10**:303–3089. Available from: http://www.naun.org/main/NAUN/ijmmas/2016/a782001-aan.pdf

[10] Condotta A, Knust S, Shakhlevich NV. Parallel batch scheduling of equal-length jobs with release and due dates. Journal of Scheduling. 2010;**13**:463–477

[11] Del Ser, Javier (Ed.). A harmony search approach for the selective pick-up and delivery problem with delayed drop-off. In Harmony Search Algorithm. Berlin Heidelberg: Springer; 2016. pp. 121–131

[12] Della Croce F, T'kindt V. Improving the preemptive bound for the single machine dynamic maximum lateness problem. Operations Research Letters. 2010;**38**:589591

[13] Garey MR, Johnson DS. Computers and intractability: A guide to the theory of NP-completeness. San Francisco: Freeman; 1979

[14] Garey MR, Johnson DS, Simons BB, Tarjan RE. Scheduling unit–time tasks with arbitrary release times and deadlines. SIAM Journal on Computing. 1981;**10**:256–269

[15] Gharbi A, Labidi M. Jackson's semi-preemptive scheduling on a single machine. Computers & Operations Research. 2010;**37**:2082–2088

[16] Glover F. Tabu-search: A tutorial. Interfaces. 1990;**20**:74–94

[17] Grabowski J, Nowicki E, Zdrzalka S. A block approach for single-machine scheduling with release dates and due dates. European Journal of Operational Research. 1986;**26:**278–285

[18] Graham RL, Lawler EL, Lenstra JL, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics. 1976;**5**:287–326

[19] Hall LA, Shmoys DB. Jacksons rule for single-machine scheduling: Making a good heuristic better. Mathematics of Operations Research 1992;**17**:22–35

[20] Ivens P, Lambrecht M. Extending the shifting bottleneck procedure to real-life applications. European Journal on Operations Research. 1996;**90**:252–268

[21] Jackson JR. Scheduling a production line to minimize the maximum tardiness. Los Angeles, CA: Management Science Research Project, University of California; 1955

[22] Kirkpatrick S, Gelant CD, Vecchi MP. Optimization by simulated annealing. Science. 1983;**220**:924–928

[23] Lawton G. Genetic algorithms for schedule optimization. AI Expert. 1992;23–27

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

[25] Ow PS, Morton TE. Filtered beam search in scheduling. International Journal of Production Research. 1988;**26**:35–62

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

[27] Potts CN. Analysis of a heuristic for one machine sequencing with release dates and delivery times. Operations Research. 1980;**28:**1436–1441

[28] Toth P, Vigo D, editors. Vehicle Routing Problem (SIAM Monographs On Discrete Mathematics and Applications, vol. 386). Philidelphia, PA: SIAM; 2002

[29] Schutten JMJ. Practical job shop scheduling. Annals of Operations Research. 1998;**83**: 161–177

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

[31] Simons B. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. SIAM Journal of Computing. 1983;**12**:294–299

[32] Simons B, Warmuth M. A fast algorithm for multiprocessor scheduling of unit-length jobs. SIAM Journal of Computing. 1989;**18**:690–710

[33] Vakhania N. A better algorithm for sequencing with release and delivery times on identical processors. Journal of Algorithms. 2003;**48**:273–293

[34] Vakhania N. Single-machine scheduling with release times and tails. Annals of Operations Research. 2004;**129**:253–271

[35] Vakhania N. Scheduling jobs with release times preemptively on a single machine to minimize the number of late jobs. Operations Research Letters. 2009;**37**:405–410

[36] Vakhania N. Branch less, cut more and minimize the number of late equal-length jobson identical machines. Theoretical Computer Science. 2012;**465**:49–60

[37] Vakhania N. A study of single-machine scheduling problem to maximize throughput. Journal of Scheduling. 2013;**16**(4):395–403

[38] Vakhania N, Shchepin E. Concurrent operations can be parallelized in scheduling multiprocessor job shop. Journal of Scheduling. 2002;**5**:227–245

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

[40] Vakhania N, Perez D, Carballo L. Theoretical expectation versus practical performance of Jackson's Heuristic. Mathematical Problems in Engineering. 2015;**2015**: ID 484671. DOI: http://dx.doi.org/10.1155/2015/484671

[41] Vakhania N, Hernandez JA, Alonso-Pecina F, Zavala C. A Simple heuristic for basic vehicle routing problem. Journal of Computer Science Technology Updates. 2016;**3**(2):38–44. DOI: http://dx.doi.org/10.15379/2410-2938.2016.03.02.04

IntechOpen