An Efficient Heuristic for a Discrete Optimization Problem

Nodari Vakhania*, Elisa Chinos and Crispin Zavala

Centro de Investigación en Ciencias, UAEMor, Mexico

Abstract: In this paper we deal with a discrete optimization problem, which, among many other such problems, is computationally intractable. Since the existence of an exact solution algorithm for our problem is highly unlikely, the development of heuristic and approximation algorithms is of a great importance. Here we briefly discuss this issue and describe a robust 2-approximation heuristic that is used for getting an approximation solution for the problem of scheduling jobs with release times and due-dates on a single machine to minimize the maximum job lateness.

Keywords: Discrete optimization, Feasible solution, Scheduling, Heuristic, Approximation algorithm.

INTRODUCTION

It is common in our everyday life a desire to optimize time, to do our duties in a due time or to arrange as many things as possible in a limited time. Sometimes optimization of our time is a purely personal matter, but often it depends on the factors and the environment around us. We wish to arrange them in a way which would be beneficial for us. As an example we give a problem which many of us encounter daily. A family with *n* members lives in an apartment with a single bathroom, shower, iron and a mirror. Each member of the family daily gets up at a fixed time, needs a prescribed time units in the bathroom, in the shower, in front of the mirror and he (she) also needs the iron for the prescribed time units (to arrange the cloths to ware on this day). The bathroom, shower, iron and the mirror are our *resources* and can be used by at most one person at a time. Since each member of the family has to leave home at a prescribed time (in order the reach the job/school on time), we face an optimization problem, how to arrange the activities of each member of the family on each of the above resources to minimize the overall time needed for all members of the family to complete all activities. These types of problems are dealt with in *discrete optimization*, in particular, in *scheduling theory*.

Discrete optimization (DO) problems constitute a significant class of practical problems with a discrete nature. They have emerged in late 40-s of 20th century. With a rapid grow of the industry, the new demands in the optimal solution of the newly emerged resource management and distribution problems have a risen. For the development of effective solution methods, these problems were formalized and addressed mathematically.

DO PROBLEMS AND APPROXIMATION

A DO 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, *i.e*., 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 essentially changed if in the future, much faster computers will be developed.

The DO 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 the amount of the computer memory necessary to represent the problem data/parameters). Furthermore, all *NP*-hard problems, ones from the second class, 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

*Address correspondence to this author at the Centro de Investigación en Ciencias, UAEM or, Mexico; Tel: 52 777 329 70 20;
Fax: 52 777 329 70 40; E-mail: nodari@uaem.mx

E-ISSN: 2410-2938/15

algorithm for any other *NP*-hard problem. At the same time, it is believed that it's 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*, such an algorithm is highly unlikely to exist for an *NP*-hard problem. Hence, one chooses a compromise to solve such a problem approximately. *Heuristic* algorithms can be used for the exact solution of a problem in class *P* and for an approximate solution of an *NP-hard* problem. A heuristic algorithm is an efficient polynomial-time algorithm that creates one or more feasible schedules. A *greedy* algorithm is a heuristic that typically works on *n* (external) iterations, were *n* is the number of objects in the given problem. Since the size of a discrete optimization problem is a polynomial in *n*, 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, *i.e*., no greedy or heuristic algorithm can solve optimally an *NP*-hard problem (unless *P* = *NP*, which is very unlikely). Since the majority of DO 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*, *i.e*., ones which deliver a solution with a guaranteed deviation from an optimal one in polynomial time. Since the simplest polynomial-time algorithms are greedy, a greedy algorithm is a simplest approximation algorithm.

The *performance ratio* of an approximation algorithm *A* measures the quality of this approximation algorithm. It is the ratio of the value of the objective function delivered by algorithm *A* to the optimal value. A *κ-approximation algorithm* is one with the worst-case performance ratio *κ*. Another commonly used measure is the *absolute error* which is just the difference between the value of the objective function delivered by algorithm *A* and the optimal objective value.

SCHEDULING PROBLEMS

Heuristic and approximation algorithms are common and important, in particular, for the *scheduling problems*. These problems deal with a finite set of requests called *jobs* to be performed (or scheduled) on a finite (and limited) set of resources called *machines* (or *processors*). The aim is to choose the order of processing the jobs on machines so as to meet a given objective criteria. In our family example, to go to the bathroom, to take a shower, to iron and to brush the hair in front of the mirror are examples of requests; the bathroom, shower, iron and mirror are examples of resources. A job in a factory or a program in a computer system or a lesson at school are other examples of requests. A machine in a factory or a processor in a computer system or a teacher in a school are other examples of resources. Each job has its *processing time*, *i.e*., it needs a prescribed time on a machine, and a machine cannot handle more than one request at a time (for example, a teacher cannot give two different lessons simultaneously). Besides, there is a limited number of machines (which are expensive in use) and time is limited. We need to arrange an order in which the jobs are handled by the machines to minimize or maximize some important, often time, criterion or *objective functions*.

A scheduling problem might be a single-stage, for instance, single-machine or multiprocessor scheduling problems, or a multi-stage shop scheduling problem. There are three basic type of multiprocessor scheduling problems with *identical*, *uniform* and *unrelated* parallel processor environments. The first two are characterized by an operation-independent speed function (for identical machines every machine has the same speed). A group of unrelated machines has no uniform speed characteristic, *i.e*., a machine speed is operation-dependent. There are also three basic shop scheduling problems, which are *open-shop, flow-shop* and *job-shop* scheduling problems.

More formally, a multiprocessor is a triple constituted by the sets of jobs J and machines M and a *processing time function f*, a mapping from J × M to *R*, where the value of this function for a pair *J,M* is the *processing time (length)* of job *J* on machine *M* denoted by *M*(*J*). A multiprocessor without any restriction on its processing time function is called a system of *unrelated processors*. In a system of

identical processors for each P and Q from M and for each $J \in$ J, $P(J) = Q(J)$.

There are three basic shop scheduling problems. We let J,M stand for a shop scheduling instance with the set of jobs J = $\{J_1,...,J_n\}$ and the set of machines M = $\{M_1,...,M_m\}$. In an instance of the *job-shop* $J//C_{\max}$ each job from J is an ordered set of elements called *operations*. [1] Each operation is to be scheduled on one particular machine from M. J_i^j is the operation of job J^j to be performed on machine M_i (we shall deal with job-shops in which every job has no more than one operation to be scheduled on one particular machine). We will write $J_i^j \rightarrow J_k^j$ if J_i^j *immediately precedes* J_k^j according to the operation order in J^j. Operation J_i^j has a *processing time* or *length* p_i^j, which is the amount of time it takes on machine M_i. J_i^j is a *dummy operation* of job J^j on machine M_i if $p_i^j = 0$.

The *open-shop* $O//C_{\max}$ is a special case of the job-shop in which there is no precedence order between the operations of any job, these operations can be processed in an arbitrary order on their corresponding machines. The *flow-shop* $F//C_{max}$ is another special case of job-shop scheduling problem in which the operation order in all jobs is the same, *i.e*., every job is processed by the machines in the same predetermined order.

JACKSON'S HEURISTIC FOR SCHEDULING JOBS WITH RELEASE TIMES AND DUE-DATES

One of the oldest and commonly used (on-line) heuristics in scheduling theory is that of Jackson [14]. 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 [8]). As we will see a bit later, in the worst-case, Jackson's heuristic will deliver a solution which is twice worse than an optimal one.

THE PROBLEM FORMULATION

In our single-machine scheduling problem $1|r_j|L_{\max}$ (Graham *et al*. [12]) we are given n jobs in $\{1,2,...,n\}$. Each job j has (non-interruptible) *processing time* p_j, *release time* r_j and *due-date* d_j. The n jobs are to be scheduled on a single machine that can process at most one job at a time. The release time of job j, r_j, is the time moment when job j arrives to the system hence becomes available for processing on the machine, whereas its due-date d_j is the desired completion time for job j.

A *feasible schedule* S is a mapping that assigns to each job j a starting time $t_j(S)$, such that $t_j(S) \geq r_j$ and $t_j(S) \geq t_k(S) + p_k$, for any job k included earlier in S (for notational simplicity, we use S also for the corresponding jobset); the first inequality says that a job cannot be started before its release time, and the second one reflects the restriction that the machine can handle only one job at any time. $c_j(S) = t_j(S) + p_j$ is the completion time of job j. We aim to find out if there is a schedule which meets all job due-dates, *i.e*., every j is completed by time d_j. If there is no such schedule then we look for an optimal schedule, *i.e*., one minimizing the maximum job *lateness* $L_{\max} = \max\{j|c_j - d_j\}$. We denote by $L(S)$ ($L_j(S)$, respectively) for the maximum lateness in S (the lateness of job j in S, respectively).

There is an equivalent formulation of the above problem in which the due-dates are replaced by the *delivery times* and the maximum job completion time is minimized. In this setting, n jobs have to be scheduled on a single machine. Each job j again 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). Here 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 (Graham *et al*. [12]) this version 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

[1] We use the standard three-field notation for scheduling problems originally introduced in [12]

parameters, and in the third field the objective criteria is given.

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

Because of the above equivalence, one can use both above formulations interchangeably. We may just briefly mention here that 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.

DESCRIPTION OF THE HEURISTIC AND SOME RELATED WORK

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). In more details, the heuristic distinguishes n scheduling times, the time moments at which a job is assigned to the machine. Initially, the earliest scheduling time is set to the minimum job release time. Among all jobs released by that time a job with the minimum due-date (the maximum delivery time, alternatively) is assigned to the machine (ties being broken by selecting a longest job). Iteratively, the next scheduling time is either the completion time of the latest assigned so far job to the machine or the minimum release time of a yet unassigned job, whichever is more (as no job can be started before the machine gets idle nether before its release time). And again, among all jobs released by this scheduling time a job with the minimum due-date (the maximum delivery time, alternatively) is assigned to the machine. Note that the heuristic creates no gap that can be avoided always scheduling an already released job once the machine becomes idle, whereas among yet unscheduled jobs released by each scheduling time it gives the priority to a most urgent one (*i.e*., one with the smallest due-date, alternatively, with the 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 [17] 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 [13] 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 [17]. Jackson's heuristic can be efficiently used for the solution of shop scheduling problems. Using Jackson's heuristic as a schedule generator, McMahon & Florian [15] and Carlier [3] have proposed efficient enumerative algorithms for $1|r_j,q_j|C_{\max}$. Grabowski *et al.* [11] use the heuristic for the obtention of an initial solution in another enumerative algorithm for the same problem. Garey *et al*. [9] 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 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 [21] 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 [22] 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 [24] and [23], 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 [18] and

Simons and Warmuth [19], respectively. Using the same heuristic as a schedule generator in [20] was proposed an $O(q_{\max}mn\log n+O(mvn))$ algorithm for the minimization version of the latter problem, where $q_{\max}$ is the maximal job delivery time and $v < n$ 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 [3], Carlier and Pinson [4] and Brinkkotter and Brucker [2] and more recent works of Gharbi and Labidi [10] and Della Croce and T'kindt [7]. Carlier and Pinson [5] have used the extended Jackson's heuristic for the solution of the multiprocessor job-shop problem with identical machines, and it can also be adopted for the case when parallel machines are unrelated (see [25]). Jackson's heuristic can be useful for parallelizing the computations in scheduling job-shop Perregaard and Clausen [16], and also for the parallel batch scheduling problems with release times Condotta *et al*. [6].

THE WORST-CASE BOUND

In this subsection we will see why the basic Jackson's heuristic is a 2-approximation one. 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), *i.e*., 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, respectively) denotes the starting (completion, respectively) time of job j.

A *block* in a J-schedule is its consecutive part consisting of the successively scheduled jobs without any gap in between preceded and succeeded by a (possibly a 0-length) gap.

J-schedules have useful structural properties. The following basic definitions, taken from [20], 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, *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.

A statement, similar to Lemma 1 can be found in reference [26] and Lemma 2 in [20]. Lemma 4 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^0 with $L(S^0) < L_o(S)$, *i.e*., 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 (see Observation 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 I above) the *live* emerging job.

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

Corollary 1

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

Below we use T^S for the makespan (maximum full job completion time) of Jschedule S, and T^* (L^*_{max}, respectively) for the optimum makespan (lateness, respectively).

Lemma 2

$$T^\sigma - T^* < Pl\left(L^\sigma_{\max} - L^*_{\max} < Pl\right)$$

where l is the live emerging job for kernel K ∈ σ.

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

Lemma 2 implicitly defines a lower bound of $T^\sigma - p_I$ derived from the solution of the non-preemptive Jackson's heuristic. This lower bound can further be strengthen using the following concept. Let the delay for kernel $K \in \sigma$, $\delta(K,I)$ be $c_I - r(K)$ (I (o, respectively) stand again for the live emerging (overflow, respectively) job for kernel K).

Lemma 3

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

The proof is similar to that of Lemma 2, with an extra observation that the delay for the earliest scheduled job of kernel K is defined more accurately by $\delta(K,I)$.

Observe that $\delta(K,I) < p_I$, and, in paractice, $\delta(K,I)$ can be drastically smaller than p_I.

We can easily derive a 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 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 for $K \in \sigma$ then σ is optimal by Corollary 1. Suppose I exists; clearly, $p_I < T^*$ (as job I has to be scheduled in S^* and there is at least one more (kernel) job in it). Then by Lemma 2,

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

A BETTER PRACTICAL BEHAVIOR

The above worst-case bound of 2 might be too rough in practice, when the solution quality is important; *i.e*., solutions with the objective value better than twice the optimal objective value are required. In addition, the solutions may need to be created on-line (any possible off-line modification of the heuristic that may lead to a better performance would be of not much use). In this situation, the on-line performance measure is essentially important.

As it was shown in [27] the quality of the solution delivered by the basic Jackson's heuristic is essentially related with the maximum job processing time $p_{\max}$ that may occur in a given problem instance. In particular, the interference of a "long" non-urgent job with the following scheduled urgent jobs affects the solution quality.

In [27] $p_{\max}$ is expressed as a fraction the optimal objective value and a much more accurate approximation ratio than 2 is derived. In some applications, this kind of relationship can priory 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 may 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 in [27] an explicit expression of the heuristic's approximation ratio in terms of that fraction is derived. In particular, it is shown that Jackson's heuristic will always deliver a solution within a factor of $1 + 1/\kappa$ of the optimum.

The above estimations may drastically outperform the earlier known wors-tcase ratio of 2, in practice. Due to the computational experiments reported in [27], 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 the 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.

REFERENCES

[1] Bratley P, Florian M, Robillard P. On sequencing with earliest start times and due–dates with application to computing bounds for (n/m/G/Fmax) problem. Naval Res. Logist. Quart 1973; 20: 57-67. http://dx.doi.org/10.1002/nav.3800200106

[2] Brinkkotter W, Brucker P. Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments. J of Scheduling 2001; 4: 53-64. http://dx.doi.org/10.1002/1099-1425(200101/02)4:1<53::AID-JOS59>3.0.CO;2-Y

[3] Carlier J. The one–machine sequencing problem. European J of Operations Research 1982; 11: 42-47 http://dx.doi.org/10.1016/S0377-2217(82)80007-6

[4] Carlier J, Pinson E. An Algorithm for Solving Job Shop Problem. Management Science 1989; 35: 164-176 http://dx.doi.org/10.1287/mnsc.35.2.164

[5] Carlier J, Pinson E. Jakson's pseudo preemptive schedule for the Pm/ri,qi/Cmax problem. Annals of Operations Research 1998; 83: 41-58. http://dx.doi.org/10.1023/A:1018968332237

[6] 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. http://dx.doi.org/10.1007/s10951-010-0176-y

[7] Della Croce F, T'kindt V. Improving the preemptive bound for the single machine dynamic maximum lateness problem. Operations Research Letters 2010; 38: 589-591. http://dx.doi.org/10.1016/j.orl.2010.08.002

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

[9] Garey MR, Johnson DS, Simons BB, Tarjan RE. Scheduling unit–time tasks with arbitrary release times and deadlines. SIAM J Comput 1981; 10: 256-269. http://dx.doi.org/10.1137/0210018

[10] Gharbi A, Labidi M. Jackson's Semi-Preemptive Scheduling on a Single Machine. Computers & Operations Research 2010; 37: 2082-2088 http://dx.doi.org/10.1016/j.cor.2010.02.008

[11] Grabowski J, Nowicki E, Zdrzalka S. A block approach for singlemachine scheduling with release dates and due dates. European J. of Operational Research 1986; 26: 278-285. http://dx.doi.org/10.1016/0377-2217(86)90191-8

[12] Graham RL. Lawler EL, Lenstra JL, Rinnooy AHG. Kan. Optimization and approximation in deterministic sequencing and scheduling: a servey. Ann Discrete Math 1979; 5: 287-326. http://dx.doi.org/10.1016/S0167-5060(08)70356-X

[13] Hall LA, Shmoys DB. Jackson's rule for single-machine scheduling: Making a good heuristic better, Mathematics of Operations Research 1992; 17: 22-35 http://dx.doi.org/10.1287/moor.17.1.22

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

[15] McMahon G, Florian M. On scheduling with ready times and due dates to minimize maximum lateness. Operations Research 1975; 23: 475-482. http://dx.doi.org/10.1287/opre.23.3.475

[16] Perregaard M, Clausen J. Parallel branch-and-bound methods for the job-shop scheduling problem. Annals of Operations Research 1998; 83: 137-160. http://dx.doi.org/10.1023/A:1018903912673

[17] Potts CN. Analysis of a heuristic for one machine sequencing with release dates and delivery times. Operations Research 1980; 28: 1436-1441. http://dx.doi.org/10.1287/opre.28.6.1436

[18] Simons B. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. SIAM J. Comput 1983; 12: 294-299. http://dx.doi.org/10.1137/0212018

[19] Simons B, Warmuth M. A fast algorithm for multiprocessor scheduling of unit-length jobs. SIAM J. Comput 1989; 18: 690-710. http://dx.doi.org/10.1137/0218048

[20] Vakhania N. A better algorithm for sequencing with release and delivery times on identical processors. Journal of Algorithms 2003; 48: 273-293 http://dx.doi.org/10.1016/S0196-6774(03)00072-5

[21] Vakhania N. Single-Machine Scheduling with Release Times and Tails. Annals of Operations Research 2004; 129: 253-271. http://dx.doi.org/10.1023/B:ANOR.0000030692.69147.e2

[22] 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. http://dx.doi.org/10.1016/j.orl.2009.09.003

[23] Vakhania N. Branch less, cut more and minimize the number of late equal-length jobs on identical machines. Theoretical Computer Science 2012; 465: 49-60 http://dx.doi.org/10.1016/j.tcs.2012.08.031

[24] Vakhania N. A study of single-machine scheduling problem to maximize throughput. Journal of Scheduling 2013; 16(4): 395-403. http://dx.doi.org/10.1007/s10951-012-0307-8

[25] Vakhania N, Shchepin E. Concurrent operations can be parallelized in scheduling multiprocessor job shop, J. Scheduling 2002; 5: 227-245. http://dx.doi.org/10.1002/jos.101

[26] 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. http://dx.doi.org/10.1016/j.tcs.2013.07.001

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

Received on 05-12-2015 Accepted on 31-12-2015 Published on 05-01-2016

http://dx.doi.org/10.15379/2410-2938.2015.02.02.05