Proceedings of the 16th Conference on Computer Science and Intelligence Systems pp. 221–225
DOI: 10.15439/2021F66
ISSN 2300-5963 ACSIS, Vol. 25

Worst-Case Analysis of an Approximation Algorithm for Single Machine Scheduling Problem

Natalia Grigoreva
St.Petersburg State University
Universitetskay nab. 7/9, St.Petersburg, Russia
Email: n.s.grig@gmail.com

***Abstract*—The problem of minimizing the maximum delivery times while scheduling jobs on the single processor is a classical combinatorial optimization problem. This problem is denoted by $1|r_j, q_j|C_{\max}$, has many applications, and it is NP-hard in strong sense. The goal of this paper is to propose a new 3/2-approximation algorithm, which runs in $O(n \log n)$ time. We proved that the bound of 3/2 is tight. To check the efficiency of the algorithm we tested it on random generated problems of up to 5000 jobs.**

Keywords: single-machine scheduling problem, release and delivery times, approximation algorithm, worst-case performance ratio

I. Introduction

WE CONSIDER a set of jobs $U = \{1, 2, \ldots, n\}$. Each job i must be processed without interruption for $t_i > 0$ time units on the processor, which can process at most one job at time. Each job i has a release time $r_i \geq 0$, when the job is ready for processing, and a delivery time $q_i \geq 0$. The delivery of each job begins immediately after processing has been completed.The objective is to minimize the time, by which all jobs are delivered. In the notation of Graham *et al.*[5] this problem is denoted by $1|r_j, q_j|C_{\max}$, and has many applications.

It is required to construct a schedule, that is, to find for each job $i \in U$ the start time τ_i, provided that $r_i \leq \tau_i$. The goal is to construct a schedule that minimizes $C_{\max} = \max\{\tau_i + t_i + q_i | i \in U\}$, which is the delivery time of the last job.

In [11] it is shown that the problem is NP-hard in the strong sense, but there are exact polynomial algorithms for some special cases. Some authors considered an equivalent formulation of the problem, in which instead of the delivery time for each job, the due date $D_i = K - q_i$, is known, where K is a constant, and the objective function is the maximum lateness $L_{\max} = \max\{\tau_i + t_i - D_i | i \in U\}$. This formulation of the problem is denoted as $1|r_i|L_{\max}$. The advantage of the model with delivery times is that the value of the objective function is always positive, while the maximum lateness can be negative or equal to zero.

If we swap the delivery times and the release times, we get an inverse problem with the property that the solution of the direct problem $S = (i_1, i_2, \ldots, i_n)$ is optimal if and only if the permutation $S_{inv} = (i_n, i_{n-1}, \ldots, i_1)$ is the optimal solution of the inverse problem.

The $1|r_j, q_j|C_{\max}$ is the main subproblem in many important models of scheduling theory, such as flowshop and job-shop problems, multiprocessor scheduling and online single-machine scheduling [12].The study of this problem is of theoretical interest and is useful in practical industrial application [1], [4], [18].

Several approximation algorithms are known for solving the problem $1|r_i, q_i|C_{\max}$.

The first algorithm for constructing an approximation schedule is the Schrage heuristic [17] - an extended Jackson rule, which is formulated as follows: each time the processor is free, a ready job with the maximum delivery time is assigned to it. Computational complexity of the Schrage heuristic is $O(n \log n)$, the algorithm is a 2-approximation algorithm [10].

K. Potts [16] proposed an algorithm in which the extended Jackson's rule algorithm repeats n times. Computational complexity of the Potts algorithm is $O(n^2 \log n)$. The worst-case performance ratio is equal 3/2.

L. Hall and D. Schmois [8] have developed the method in which the Potts algorithm is applied to the direct and inverse problem. In total, the algorithm builds $4n$ schedules and chooses the best one. Computational complexity of the algorithm is $O(n^2 \log n)$. The worst-case performance ratio is equal 4/3.

E. Novitsky and K. Smutnitsky [14] proposed an 3/2-approximation algorithm, which creates only two permutations. For the first time, the Jackson rule is applied, then the interference job is determined and the set of jobs is divided into two sets: jobs that should be performed before the interference job in the order of their release times, and after it that should be performed after it in non-increasing delivery times. The best schedule is selected from two schedules. Computational complexity is $O(n \log n)$.

All the mentioned algorithms use the list greedy Schrage algorithm as a basic heuristic.

The works of [2], [3], [13], [15] developed branch and bound algorithms for single processor scheduling problem using different branching rules and bounding techniques. The first efficient algorithm is Carlier algorithm [3], which optimally solves instances with up to thousand of jobs. This algorithm constructs a full solution by extended Jackson's rule in each node of the search tree.

One way to improve the performance of the branch and bound method is to use approximation efficient algorithms to obtain upper bounds. Such algorithms should have a good approximation ratio and the low computational complexity.

One of the popular scheduling tools are list algorithms that build non-delayed schedules. In the list algorithm, at each step, the job with the highest priority is selected from the set of ready jobs. But the optimal schedule may not belong to the class of non-delayed schedules.

IIT schedules (IIT - inserted idle time) were defined in [9] as feasible schedules in which the processor can be idle when there are jobs ready to run.

The author considered the IIT 2-approximation algorithm [6] and developed the branch and bound algorithm for single-machine scheduling problem with receive and delivery times [7]. The main idea of greedy algorithms for solving this problem is the choice at each step of the highest priority job, before the execution of which the processor could be idle.

In this paper, we propose an approximation algorithm ICA for solving the problem, which creates two permutations, one by the Schrage method, and the second by an algorithm with inserted idle time. The construction of each permutation requires $O(n \log n)$ action.

The article is organized as follows: Section 2 presents a new approximate algorithm scheduling IJR and the combined ICA algorithm, which builds two permutations and chooses the best one. We prove that the worst-case performance ratio of the ICA algorithm is equal 3/2 and this bound is tight in section 3. The results of the computational experiment, which showed the speed and practical accuracy of the algorithm, are given in Section 4. In conclusion, the main results obtained in the article are formulated.

II. IJR AND ICA SCHEDULING ALGORITHMS.

First, we describe the IJR scheduling algorithm.

The main idea of the IJR algorithm is that sometimes it is better to place a priority job on service, even if it leads to some idle time of the processor.

In the IJR algorithm two jobs are selected: the highest priority job and the highest priority from ready jobs. The paper has established special conditions in which it is advantageous to organize the unforced idle time of the processor. These conditions allow to choose between two jobs.

The algorithm IJR is a greedy algorithm, but not a list algorithm and can be used as a basic heuristic for various scheduling models and constructing a branch and bound method.

We introduce the following notation: $S_k = (i_1, i_2, \ldots, i_k)$ is the partial schedule, $time = \max\{\tau_i + t_i | i \in S_{k-1}\}$ is the time to release the processor after the execution of already scheduled jobs. We store ready jobs in the queue with priorities Q_1, the priority of a job is it's delivery time.

A. Algorithm IJR

1) Sort all jobs in non-descending order of release times: $r_{j_1} \le r_{j_2} \le \ldots \le r_{j_n}$.
2) Define the lower bound of the objective function
$LB_1 = \min\{r_i |\ i \in U\} + \sum_{i=1}^{n} t_i + \min\{q_i \mid i \in U\}$.
$LB_2 = \max\{r_i + t_i + q_i \mid i \in U\}$.
$LB = \max\{LB_1, LB_2\}$.

Algorithm 1 IJR algorithm (the main loop)

1: Initialize: $S_0 = \emptyset$; $Q_1 = \emptyset$; $l \leftarrow 1$;
2: **for** $k \leftarrow 1$ to n **do**
3: **if** $Q_1 = \emptyset$; **then**
4: $time \leftarrow r_{j_l}$;
5: **end if**
6: **while** $r_{j_l} \le time$ **do**
7: Add ready job j_l to the Q_1 queue; $l \leftarrow l + 1$;
8: **end while**
9: Select the ready job $u \in Q_1$ with the maximum delivery time $q_u = \max\{q_i | i \in Q_1\}$;
10: $r_{up} \leftarrow time + t_u$;
11: **while** $r_{j_l} < r_{up}$ **do**
12: **if** $(q_{j_l} \ge LB/2) \& (q_{j_l} - q_u \le r_{j_l} - time)$ **then**
13: Set job j_l on the processor: $S_k \leftarrow S_{k-1} \cup \{j_l\}$; $\tau(j_l) \leftarrow r(j_l)$; $time \leftarrow \tau(j_l) + t(j_l)$; $l \leftarrow l + 1$; break;
14: **else**
15: j_l is added to the queue Q_1; $l \leftarrow l + 1$;
16: **end if**
17: **end while**
18: Set job u on the processor: $S_k \leftarrow S_{k-1} \cup \{u\}$; $\tau(u) \leftarrow time$; $time \leftarrow \tau(u) + t(u)$; delete u from Q_1;
19: **end for**
20: The schedule S_n and its makespan is equal $C_{\max}(S_n)$.

B. Combined scheduling algorithm ICA

1. Construct the schedule S_{JR} by the Schrage algorithm, denote the makespan of the schedule $C_{\max}(S_{JR})$.

2. Construct the schedule S by the IJR algorithm, denote the makespan of the schedule $C_{\max}(S)$.

3. Choose the schedule S_A with a smaller value of the objective function: $C_{\max}(S_A) = \min\{C_{\max}(S), C_{\max}(S_{JR})\}$.

Computational complexity of the algorithm is $O(n \log n)$. The algorithm ICA constructs two permutations: one by the Schrage algorithm, the computational complexity of which is $O(n \log n)$, and one by the IJR algorithm.

Let's show that for the IJR algorithm the computational complexity is equal $O(n \log n)$. First, we sorts all jobs in non-descending order of it's release times, this step requires $O(n \log n)$ actions.

The main operation is to select a job from a set of ready jobs. We store the ready jobs as a priority queue Q_1, which can be organized as a binary heap, the priority of job j is the delivery time q_j. At the steps 6-8 of the algorithm, we add new ready jobs to the queue Q_1 such that $r_{j_i} \le time$, adding each job requires $O(\log n)$ actions. The job u with the highest priority is selected for $O(1)$ actions (step 9).

At steps 11-17 we add new jobs j_i to the queue Q_1, for which $r_{j_i} < time + t_u$. If there is a job j_l for which all conditions (step 12) are met, then we map j_l on the processor and go to the beginning of the main cycle (step 2). Otherwise, we look through all candidates by placing them in the queue

Q_1, and map the job u on the processor on the step 18. Then the job u is deleted from queue Q_1, which requires $O(\log n)$ actions.

Each job can be added to the queue at most once: building the binary heap requires $O(n \log n)$ actions. The total computational complexity is equal $O(n \log n)$.

III. Properties of the schedule constructed by the algorithm ICA

The properties of the schedule created by the combined algorithm ICA proposed in Section 2 are formulated and proved in the following lemmas.

Let the IJR algorithm constructs a schedule S, the makespan is equal to $C_{\max}(S)$, and the schedule S_{JR} is constructed by the JR algorithm, the makespan is equal to $C_{\max}(S_{JR})$. Consider some definitions that were introduced in [16] for schedules constructed according to Jackson's rule, and which are important characteristics for IIT schedules.

Definition 3.1: [16] A critical job is a job j_c such that $C_{\max}(S) = \tau_{j_c} + t_{j_c} + q_{j_c}$. If there are several such jobs, then we choose the earliest one in the schedule S.

Definition 3.2: [16] A critical sequence in a schedule S is a sequence of jobs $J(S) = (j_a, j_{a+1}, \ldots, j_c)$ such that j_c is the critical job and there is no processor idle time in the schedule, starting from the start of the job j_a until the job j_c ends.

The job j_a is either the first job in the schedule, or the processor is idle before it.

Definition 3.3: [16] A job j_u in a critical sequence is called interference job if $q_{j_u} < q_{j_c}$ and $q_{j_i} \geq q_{j_c}$, for $i > u$.

Proposition 3.4: [16] If for all jobs of the critical sequence it is true that $r_{j_i} \geq r_{j_a}$ and $q_{j_i} \geq q_{j_c}$, then the schedule is optimal.

Let us introduce a definition of delayed job that can be encountered in IIT schedules.

Definition 3.5: A job j_v from a critical sequence $J(S) = (j_a, j_{a+1}, \ldots, j_c)$ is called a delayed job if $r_{j_v} < r_{j_a}$.

An interference job can be a delayed job.

Let us formulate two properties of the IJR schedule, similar to the properties of JR schedules [16].

Lemma 3.6: If there is the interference job j_u in a critical sequence, then $C_{\max}(S) - C_{\max}(S_{opt}) < t_{j_u}$

Lemma 3.7: If there is no any delayed jobs in the critical sequence, then $C_{\max}(S) - C_{\max}(S_{opt}) \leq q_{j_c}$.

The proof of the lemmas is similar to [16], in Lemma 3.7 it is necessary to add the condition that there are no delayed jobs.

Let us introduce the following notation: if J is some sequence of jobs, then $T(J) = \sum_{i \in J} t_i$ and $r_{\min}(J) = \min\{r_i | i \in J\}$.

Lemma 3.8: If the interference job j_u in the critical sequence $J(S) = (S_1, j_u, S_2)$ is executed after the sequence S_2 in an optimal schedule or between j_a and j_c, then $C_{\max}(S)/C_{\max}(S_{opt}) \leq 3/2$.

Proof: If $t_{j_u} \leq C_{\max}(S_{opt})/2$, then the lemma 3.8 is true by lemma 3.6.

Let $t_{j_u} > C_{\max}(S_{opt})/2$. If the interference job j_u is executed after all jobs of the sequence S_2 in an optimal schedule, then $C_{\max}(S_{opt}) \geq r_{\min}(S_2) + T(S_2) + t_{j_u} + q_{j_u}$.

Then

$$C_{\max}(S) - C_{\max}(S_{opt}) \leq r_{j_a} + T(S_1) + t_{j_u} + T(S_2) +$$
$$+ q_{j_c} - r_{min}(S_2) - T(S_2) - t_{j_u} - q_{j_u} =$$
$$= r_{j_a} + T(S_1) - r_{min}(S_2) + q_{j_c} - q_{j_u} =$$
$$= -idle + q_{j_c} - q_{j_u}.$$

Where $idle = -r_{j_a} - T(S_1) + r_{min}(S_2) > 0$. If $q_{j_c} < LB/2$, then the lemma has proven. Let $q_{j_c} \geq LB/2$. Choose a job $v \in S_2$ such that $r_v = r_{\min}(S_2)$. Then $q_v \geq LB/2$, and $idle = r_v - time > q_v - q_{j_u} \geq q_{j_c} - q_{j_u}$.

Then $C_{\max}(S) - C_{\max}(S_{opt}) < LB/2$.

If in the optimal schedule, the job j_u is performed between jobs j_a and j_c, then

$$C_{\max}(S_{opt}) \geq r_{j_a} + t_{j_u} + T(S_2) + q_{j_c}.$$

Therefore $C_{\max}(S) - C_{\max}(S_{opt}) \leq r_{j_a} + T(S_1) + t_{j_u} + T(S_2) + q_{j_c} - r_{j_a} - t_{j_u} - T(S_2) - q_{j_c} = T(S_1) < LB/2$. ■

Lemma 3.9: Let the schedule S_{JR} is constructed by the JR algorithm. There is the interference job j_u in the critical sequence $J(S_{JR}) = (F_1, j_u, F_2)$.

If the interference job j_u is executed before all jobs of the sequence F_2 in an optimal schedule, then $C_{\max}(S_{JR})/C_{\max}(S_{opt}) \leq 3/2$.

Proof: If $t_{j_u} \leq C_{\max}(S_{opt})/2$, then the lemma 3.9 is true by lemma 3.6. If the job j_u is executed before all jobs of the sequence F_2 in an optimal schedule S_{opt}, then

$$C_{\max}(S_{opt}) \geq r_{z_a} + t_{j_u} + T(F_2) + q_{z_c}.$$

Then $C_{\max}(S_{JR}) - C_{\max}(S_{opt}) \leq T(F_1) < LB/2$. ■

Theorem 3.10: The algorithm ICA constructs a schedule S_A for which $C_{\max}(S_A)/C_{\max}(S_{opt}) \leq 3/2$.

Proof: Let the schedule S be constructed using the IJR algorithm and the schedule S_{JR} be constructed by the JR algorithm. There are the critical sequence $J(S) = (j_a, j_{a+1}, \ldots, j_c)$ in S, and the critical sequence $J(S_{JR}) = (z_a, z_{a+1}, \ldots, z_c)$ in S_{JR}.

We consider all the possible cases.

Case 1. There are no interference and delayed jobs in $J(S)$ or no interference job in $J(S_{JR})$ critical sequences.

In this case, the corresponding algorithm has constructed an optimal schedule.

Case 2. There are interference jobs in each critical sequence. It is required to consider the case in which two interference jobs is the same large job such that $t_{j_u} > C_{\max}(S_{opt})/2$.

The makespan of the schedule S_{JR} is equal to $C_{\max}(S_{JR}) = r_{z_a} + T(J(S_{JR})) + q_{z_c}$.

If delivery time of the critical job z_c does not exceed $C_{\max}(S_{opt})/2$, then by Lemma 3.7 the theorem is true. Let $q_{z_c} > C_{\max}(S_{opt})/2$.

By virtue of the proved Lemmas 3.8 and 3.9, it suffices to consider the case in which in the optimal schedule, the job j_u should be carried out after all jobs of the sequence F_2 and before the job j_a.

TABLE I
RELEASE, PROCESSING AND DELIVERY TIMES OF JOBS FROM U

Job	r_i	t_i	q_i
x	ε	ε	$M-2*\varepsilon$
a	$M/2-\varepsilon$	ε	$M/2$
u	0	$M/2+\varepsilon$	0
c	$M/2+\varepsilon$	ε	$M/2-2*\varepsilon$

According to the properties of the IJR algorithm, the processor is idle until the time r_{j_a} and $q_{j_a} > LB/2$.

Then $C_{\max}(S_{opt}) \geq r_{\min}(F_2) + T(F_2) + t_{j_u} + t_{j_a} + q_{j_a}$. Hence,

$$C_{\max}(S_{JR}) - C_{\max}(S_{opt}) \leq r_{z_a} + T(J(S_{JR})) + q_{z_c} -$$

$$-r_{\min}(F_2) - T(F_2) - t_{j_u} - t_{j_a} - q_{j_a} =$$

$$= r_{z_a} + T(F_1) + q_{z_c} - r_{\min}(F_2) - t_{j_a} - q_{j_a} <$$

$$< q_{z_c} - q_{j_a} < LB/2.$$

This is true because $q_{z_c} < LB$ and $q_{j_a} > LB/2$. In this case, the Schrage algorithm constructs a 3/2 approximation schedule.

Case 3. There is the interference job in the critical sequence $J(S_{JR})$ and there are some delayed jobs in $J(S)$.

If there is no an interference job in the critical sequence $J(S)$, then $q_i \geq q_{j_c}$ for all jobs from the critical sequence $i \in J(S)$. But in the critical sequence there are jobs that can be started before the job j_a.

Then $C_{\max}(S_{opt}) \geq r(J(S)) + T(J(S)) + q_{j_c}$. Hence

$$C_{\max}(S) - C_{\max}(S_{opt}) \leq r_{j_a} + T(J(S)) + q_{j_c} - r(J(S)) -$$

$$-T(J(S)) - q_{j_c} = r_{j_a} - r(J(S)) < LB/2.$$

We have proven that the worst-case performance ratio of ICA algorithm is equal 3/2. ■

Lemma 3.11: There is an example for which the ratio $C_{\max}(S_A)/C_{\max}(S_{opt})$ tends to 3/2.

Proof: Consider a system of four jobs $U = \{x, a, u, c\}$. The data for the system of jobs are given in Table 1, where M is a constant. The lower bound for the objective function is $LB = M$.

The IJR algorithm constructs the schedule $S = (x, a, u, c)$. The processor is idle $M/2 - \varepsilon$ time units before starting the job a. The objective function is equal $C_{\max}(S) = 3/2M$. The JR algorithm constructs the schedule $S_{JR} = (u, x, a, c)$. The objective function is equal $C_{\max}(S_{JR}) = 3/2M - \varepsilon$.

The optimal schedule is $S_{opt} = (x, u, a, c)$, the value of the objective function is equal $C_{\max}(S_{opt}) = M + \varepsilon$. When ε tends to zero, the ratio $C_{\max}(S_A)/C_{\max}(S_{opt})$ tends to 3/2. ■

IV. COMPUTATIONAL EXPERIMENT

To find out the practical efficiency of the algorithm, a computational experiment was carried out. The goals of the computational experiment were comparison of the accuracy of the IJR algorithm and of the JR Schrage algorithm and comparison of the accuracy of the combined ICA algorithm with the accuracy of the NS algorithm of Novitsky and Smutnitsky using random test examples.

The initial data was generated by the method described by Carlier [3], the same method generating of test examples were used by Novitsky and Smutnitsky when they compared their proposed algorithms with Hall and Schmois and Schrage algorithms. For each job i, three integer values were chosen with uniform distribution : q_i and r_i between 1 and nK. There were chosen the values for K from 10 to 25, which were noted by Carlier as the most difficult for the problem under consideration. For each value of n and K, we considered 100 instances. Three groups of examples were considered. The processing times for each of the groups were selected from the following intervals ($t_{\max} = 50$):

1) Type A: t_j from $[1, t_{\max}]$,
2) Type B: t_j from $[1, t_{\max}/2]$, for $j \in 1 : n-1$ and t_n from $[nt_{\max}/8, 3nt_{\max}/8]$,
3) Type C: t_j from $[1, t_{\max}/3]$, for $j \in 1 : n-2$ and t_{n-1}, t_n from $[nt_{\max}/12, 3nt_{\max}/12]$.

Groups of type B contains instances with one long job and groups of type C contains instances with two long jobs.

The value of the objective function $C_{\max}$ was compared with the optimal value of the objective function C_{opt}, which was obtained by the branch and bound method [7]. In all tables, n is the number of jobs in the instance.

For tests of type A, n were changed from 50 to 5000 and for all tests the value $K = 20$ was chosen. For tests of type A, the IJR algorithm generates more optimal solutions than the NS and JR algorithms. The JR algorithm very rarely receives optimal solution. The average relative error of the solution is small for all algorithms and decreases with increasing n.The average relative error is 0.03 percent for the IJR algorithm, 0.97 percent for the JR algorithm and 0.2 percent for the NS algorithm (for $n = 50$). For $n = 5000$ the average relative error is 0.001 percent for the IJR algorithm, 0.01 percent for the JR algorithm and 0.002 percent for the NS algorithm.

The results of experiments in which we change the constant K, from 10 to 22 does not significantly affect the results of the algorithms for instances of Type A.

The theoretical analysis of the algorithms shows that the most difficult examples take place when there are one or two long jobs. Such tests were generated in groups of type B and type C.

Tables 2 and 3 show the results of comparison of algorithms for tests of type B. For these groups of tests, we considered the combined ICA, in which the best solution was chosen of the two solutions, obtained by the JR and IJR algorithms.

TABLE II
TYPE B. THE NUMBER OF OPTIMAL SOLUTIONS.

n	K	N_{IJR}	N_{JR}	N_{NS}	N_{ICA}
100	10	51	23	25	58
100	14	29	47	48	62
100	15	25	48	49	59
100	16	53	21	34	69
100	18	46	46	49	71
100	20	24	15	16	33
100	22	44	29	33	57

TABLE III
TYPE B. THE AVERAGE RELATIVE ERROR OF ALGORITHMS.

n	K	R_{IJR}	R_{JR}	R_{NS}	R_{ICA}
100	10	1.02	1.05	1.04	1.005
100	14	1.06	1.03	1.03	1.004
100	15	1.05	1.04	1.04	1.007
100	16	1.01	1.04	1.01	1.004
100	18	1.05	1.04	1.03	1.005
100	20	1.05	1.06	1.03	1.007
100	22	1.02	1.03	1.03	1.006

Columns 3—6 of Table 2 show the number of tests (in percent) for which optimal solutions were generated by algorithms IJR, JR, NS and ICA, respectively.

Table 2 show that for tests of type B the number of optimal solutions for the ICA algorithm is greater then the number of optimal solutions for the NS algorithm.

The value of the average relative error of algorithms for tests of type B are given in the Table 3. Columns 3-6 of Table 3 show the value of the average relative error $R_A = C_{\max}(S_A)/C_{opt}$ of algorithms IJR, JR , NS and ICA, respectively.

The relative error of the solution increases for all algorithms JR, IJR, NS and it is from 1 to 6 percent on average. The author's ICA algorithm has significantly more advantages. It combines the advantages of the Schrage algorithm, which does not allow unforced idle time and IJR algorithm, which allows them. The relative error of the solution for ICA algorithm is from 1.004 to 1.007 on average, but the relative error of NS algorithm is from 1.01 to 1.04.

The worst solutions for the JR algorithm had a relative error of 23 percent, for the IJR algorithm - 19 percent but for the combined ICA algorithm had only 7 percent. No test was received during testing, for which both JR and IJR algorithms have constructed a solution with large relative error. The combined algorithm generates two permutations just like the NS algorithm, but its average relative error is significantly less, and the number of optimal solutions obtained is greater.

V. CONCLUSION

The paper considers the problem of scheduling for single processor with release and delivery times. The paper proposes a new 3/2 approximation algorithm with computational complexity $O(n \log n)$, in which the priority of the job is taken into account first and processor can be idle, when certain conditions are met. The example is given, which shows that the bound of 3/2 is tight. The computational experiment has confirmed the practical efficiency of the algorithm.

REFERENCES

[1] C. Artigues and D.Feillet, "A branch and bound method for the job-shop problem with sequence-dependent setup times," *Ann. of Oper. Res.,* vol. 159, 2008, pp. 135–159, http://dx.doi.org/10,1287/opre.49.6.854.10014.

[2] K.R. Baker, Introduction to Sequencing and Scheduling. John Wiley & Son, New York, 1974.

[3] J. Carlier, "The one machine sequencing problem," *European Journal of Operational Research*, vol.11, 1982, pp.42—47, http://dx.doi.org/10.1016/s0377-2217(82)80007-6.

[4] C. Chandra, Z. Liu, J. He, T. Ruohonen, "A binary branch and bound algorithm to minimize maximum scheduling cost," *Omega* , vol. 42, 2014, pp. 9–15, http://dx.doi.org/10.1016/j.omega2013.02.005.

[5] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Ann. of Disc. Math.* ,vol. 5, no. 10, 1979, pp. 287–326, http://dx.doi.org/10.1016/S0167-5060(08)70356-X.

[6] N. Grigoreva, "Single Machine Inserted Idle Time Scheduling with Release Times and Due Dates," *Proc. DOOR2016. Vladivostoc. Russia. Sep.19-23.2016.* Ceur-WS, vol. 1623, 2016, pp. 336–343.

[7] N.S Grigoreva, "Single Machine Scheduling with Precedence Constrains, Release and Delivery times", *in Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology–ISAT 2019- Part III.* (Advances in Intelligent Systems and Computing, v. 1052), pp. 188 –198, http://dx.doi.org/10.1007/978-3-030-30443-0-17.

[8] L.A. Hall and D.B. Shmoys, "Jackson's rule for single-machine scheduling: making a good heuristic better," *Mathematics of Operations Research*, 17 (1) , 1992, pp. 22–35.

[9] J. J. Kanet and V.Sridharan, "Scheduling with inserted idle time: problem taxonomy and literature review", *Operations Research*, vol. 48, 2000, no. 1, pp. 99–110, http: //dx.doi.org/10.1287/opre.48.1.111.12453.

[10] H. Kise, T. Ibaraki and H. Mine, "Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times", *Journal of the Operations Research Society of Japan,* vol. 22, 1979, pp. 205–224.

[11] J.K. Lenstra, A.H.G. Rinnooy Kan and P.Brucker, "Complexity of machine scheduling problems," *Ann. of Disc. Math.,* 1, 1977, pp. 343–362, http://dx.doi.org/10.1016/s0167-5060(08)707-43-X.

[12] Y. Li, E. Fadda, D. Manerba, R.Tadei and O. Terzo, "Reinforcement Learning Algorithms for Online Single-Machine Scheduling", *Proceedings of the 2020 Federated Conference on Computer Science and Information Systems, M. Ganzha, L. Maciaszek, M. Paprzycki (eds)*, ACSIS, vol. 21, 2020, pp. 277–283, http://dx.doi.org/10.15439/2020F100

[13] Z. Liu, "Single machine scheduling to minimize maximum lateness subject to release dates and precedence constraints," *Computers & Operations Research*, vol. 37, 2010, pp. 1537–1543, http://dx.doi.org/10.1016/j.cor.2009.11.008.

[14] E. Nowicki and C. Smutnicki, "An approximation algorithm for a single-machine scheduling problem with release times and delivery times", *Discrete Applied Mathematics,* 48, 1994, pp. 69–79, http://dx.doi.org/10.1016/0166-218X(92)00110-8.

[15] Y. Pan, L. Shi, "Branch and bound algorithm for solving hard instances of the one-machine sequencing problem",*European Journal of Operational Research*, 168, 2006, pp. 1030–1039, http://dx.doi.org/10.1016/j.ejor.2004.07.050.

[16] C.N. Potts, "Analysis of a heuristic for one machine sequencing with release dates and delivery times", *Operations Research*, 28, 1980, pp. 1436–1441, http://dx.doi.org/10.1287/opre.28.6.1436.

[17] L. Schrage, "Optimal Solutions to Resource Constrained Network Scheduling Problems", (unpublished) 1971.

[18] K. Sourirajan and R. Uzsoy, "Hybrid decomposition heuristics for solving large-scale scheduling problems in semiconductor wafer fabrication," *J. Sched.* 10, 2007, pp. 41–65, http://dx.doi.org/10.1007/s10951-006-0325-5.