A Parallel Randomized Approximation Algorithm for Non-Preemptive Single Machine Scheduling with Release Dates and Delivery Times

Article *in* Computers & Operations Research · February 2021

DOI: 10.1016/j.cor.2021.105238

CITATION
1

READS
152

3 authors:

Hossein Badri
Amirkabir University of Technology
30 PUBLICATIONS **865** CITATIONS
SEE PROFILE

Tayebeh Bahreini
Wayne State University
19 PUBLICATIONS **409** CITATIONS
SEE PROFILE

Daniel Grosu
Wayne State University
148 PUBLICATIONS **4,047** CITATIONS
SEE PROFILE

A Parallel Randomized Approximation Algorithm for Non-Preemptive Single Machine Scheduling with Release Dates and Delivery Times

Hossein Badri[a], Tayebeh Bahreini[a], Daniel Grosu[a,1,*]

[a]*Department of Computer Science, Wayne State University, Detroit, MI, USA*

Abstract

Single machine scheduling is a very fundamental scheduling problem with extensive applications in various areas ranging from computer science to manufacturing. Also, this problem is the building block of different decomposition-based algorithms for shop scheduling problems. Most variants of the single machine scheduling problem are known to be NP-hard, and therefore, many efforts have been devoted to the development of approximation algorithms for solving them. In this paper, we design a parallel randomized approximation algorithm for the non-preemptive single machine scheduling problem with release dates and delivery times ($1|r_j, q_j|C_{max}$), where the objective is to minimize the completion time of all jobs (i.e., makespan). To evaluate the performance of the proposed algorithm, we carry out a comprehensive experimental analysis on several instances of the problem. The results indicate that the proposed parallel algorithm can efficiently solve large instances achieving significant speedup on parallel systems with multiple cores.

Keywords: Single machine scheduling, parallel algorithm, randomized polynomial-time approximation algorithm

1. Introduction

Scheduling problems have received a significant attention from researchers in the field of operations research and algorithms, resulting in a large number of publications (Graham et al. 1979, Allahverdi et al. 2008, Mnich & van Bevern 2018). There are two main reasons for such significant efforts on developing novel solution methods for these problems. First, scheduling has extensive applications in several areas such as: manufacturing systems (Nguyen et al. 2017, Fazlirad & Brennan

*Corresponding author

Email addresses: hossein.badri@wayne.edu (Hossein Badri), tayebeh.bahreini@wayne.edu (Tayebeh Bahreini), dgrosu@wayne.edu (Daniel Grosu)

[1]Address: 5057 Woodward Ave, Detroit, MI 48202

2018), appointment scheduling (Gupta & Denton 2008, Marynissen & Demeulemeester 2019), operating theatre scheduling (Augusto et al. 2010, Cardoen et al. 2010), nurse scheduling (Schoenfelder et al. 2020), and many other types of problems related to capacity planning (Hall et al. 2012). Resource planning in distributed systems is another area of applications for scheduling (Mashayekhy et al. 2015, Bittencourt et al. 2017).

Second, most of the scheduling problems are known to be computationally complex (Lenstra et al. 1977). More specifically, most of the scheduling problems are NP-hard, that is, there is no algorithm to solve these problems in polynomial time, unless P=NP. The computational complexity of scheduling problems motivated researchers to devote their efforts on developing fast solution methods including heuristic methods (Kurz & Askin 2001, Weng & Lu 2005, Tsai et al. 2014). Heuristic methods are usually efficient in terms of their execution time, while their main drawback is the lack of any guarantees for the quality of solutions. Given this fact, many efforts were devoted to the development of approximation algorithms. An α-approximation algorithm is a polynomial time algorithm that, for all instances of the problem, finds a solution whose distance from the optimal solution is within a factor of α (the so-called approximation ratio) (Williamson & Shmoys 2011, Vazirani 2013).

Alongside the development of multi-core systems, researchers have attempted to leverage these systems and design fast parallel algorithms to solve scheduling problems. Most of these efforts have been devoted to developing parallel evolutionary algorithms for scheduling problems (Funabiki & Takefuji 1993, Taillard 1994, Kalashnikov & Kostenko 2008, Dulebenets 2019, Feng et al. 2017). When it comes to approximation algorithms, the majority of research has been focused on designing sequential approximation algorithms (Williamson & Shmoys 2011). In recent years, very few research works have focused on designing parallel approximation algorithms for scheduling problems (Li et al. 2018, Ghalami & Grosu 2017, 2019). Even for other NP-hard combinatorial optimization problems, very few efforts were directed at designing parallel approximation algorithms (Blelloch et al. 2011, Blelloch & Tangwongsan 2010, Rajagopalan & Vazirani 1998).

Among scheduling problems, single machine scheduling has received a significant attention, due to its widespread applications (Talebi et al. 2009, Laalaoui & M'Hallah 2016, Niu et al. 2019, Gahm et al. 2019, Shen & Zhu 2020, Zhao et al. 2020). In this paper, we design a parallel approximation algorithm for non-preemptive single machine scheduling with release dates and delivery times,

where the objective is to minimize the makespan. To the best of our knowledge, no other research has addressed the design of parallel approximation algorithms for this problem. This problem is denoted by $1|r_j, q_j|C_{max}$ (Lawler et al. 1993) and is defined as follows:

> $1|r_j, q_j|C_{max}$ *Problem*: We are given a set of n jobs that need to be scheduled on a single machine. Each job, j, $j = 1 \ldots, n$, is characterized by its processing time $p_j > 0$, release date $r_j \geq 0$, and delivery time $q_j \geq 0$. At most one job can be processed at a time, a job cannot be processed before the release date, and a job's delivery begins immediately after its processing is completed. Once a job is assigned for execution it cannot be preempted. The objective is to minimize the makespan (i.e., the time by which all jobs are delivered).

This problem is equivalent to the single machine scheduling problem with release dates and due dates, where the objective is to minimize the maximum lateness ($1|r_j|L_{max}$) (Graham et al. 1979, Nowicki & Smutnicki 1994). Algorithms for solving this problem have been widely employed as building blocks in the design of algorithms for shop scheduling problems (Adams et al. 1988, Balas et al. 1995, Carlier & Pinson 1989). This problem is strongly NP-hard (Lenstra et al. 1977), and therefore, a polynomial time approximation scheme (PTAS) is the best approximation achievable in polynomial running time, unless $P = NP$ (Garey & Johnson 1978). A PTAS is a family of $(1 + \epsilon)$-approximation algorithms whose time complexity can depend arbitrarily on $1/\epsilon$, where ϵ is a non-negative parameter that dictates how far from the optimal is the solution obtained by the PTAS (Williamson & Shmoys 2011). Our goal in this paper is to design a practical parallel randomized approximation algorithm for $1|r_j, q_j|C_{max}$, that exploits the huge processing power of modern multi-core parallel systems.

2. Related work

Schrage (1971) was the first to investigate the design of approximation algorithms for $1|r_j, q_j|C_{max}$. He designed a constant approximation algorithm which is known as the Extended Jackson's Rule (EJR). The basic idea in EJR is to schedule greedily/iteratively the job with the maximum delivery time among all schedulable jobs (whose release dates have arrived). Kise et al. (1979) proved that Schrage's algorithm is a 2-approximation algorithm. An implementation of this algorithm requiring $O(n \log n)$ running time was proposed by Carlier (1982). The EJR algorithm was employed

by other researchers as a building block of more efficient approximation algorithms (Potts 1980, Nowicki & Smutnicki 1994).

Potts (1980) designed a 3/2-approximation algorithm with $O(n^2 \log n)$ running time. The design of this algorithm is based on the concepts of critical sequence and interference job. The job j_c which is the last completed (delivered) job is called the critical job, and the critical sequence is defined as $\{j_m, \ldots, j_c\}$, where j_m is the earliest scheduled job such that the processing of each job begins immediately after the processing of the previous job in the sequence is completed (i.e., there is no idle time). An interference job j_b is a job in the critical sequence with $q_{j_b} < q_{j_c}$ which has an earlier release date than the critical job ($r_{j_b} < r_{j_c}$). While there exists an interference job, Potts's algorithm executes EJR, iteratively. In each iteration, the critical job is shifted backward in the sequence by setting $r_{j_b} = r_{j_c}$.

Nowicki & Smutnicki (1994) designed a 3/2-approximation algorithm with $O(n \log n)$ running time. This algorithm first runs Schrage's algorithm and obtains a sequence which is denoted by π^S. If there is no interference job, π^S is considered the output sequence. If there exists an interference job j_b, jobs are divided into two sets, A and B. Set A consists of jobs that satisfy $r_i \geq q_i$, and are sorted in order of nondecreasing r_i. Set B consists of jobs that satisfy $r_i < q_i$ and are sorted in order of nonincreasing q_i. The interference job is not included in any of the two sets, and the resulting permutations of set A and B are denoted by π_A and π_B, respectively. A new permutation is created as $\pi^{AB} := \pi_A r_b \pi_B$. Finally, the permutation which has the minimum makespan is chosen as the output sequence; i.e., $\min\{C_{max}(\pi^S), C_{max}(\pi^{AB})\}$.

Hall & Shmoys (1992) proposed a constant approximation algorithm for the problem with precedence constraints and two polynomial time approximation schemes (PTAS) for the problem without precedence constraints. Similarly to the previously mentioned approximation algorithms, the constant approximation algorithm (called HS1 here) is based on EJR. The constant approximation algorithm was developed based on the properties of the inverse problem. The inverse problem is obtained by interchanging r_j and q_j for all jobs and inverting the precedence relation. According to the definition of the inverse problem, $\{j_1, \ldots, j_{n-1}, j_n\}$ is an optimal sequence for the original problem if and only if $\{j_n, j_{n-1}, \ldots, j_1\}$ is an optimal sequence for the inverse problem. This algorithm has $O(n^2 \log n)$ running time and produces a sequence which is within 4/3 of the optimal solution. They also proposed an approximation scheme (called HS2 here) based on a dynamic programming

formulation whose running time is $O(16^{1/\epsilon}(n/\epsilon)^{3+4/\epsilon})$. The second approximation scheme that was proposed by the same authors (called HS3 here) is based on the idea of dividing jobs into two sets of large and small jobs, such that the number of large jobs is fixed according to the value of ϵ. The running time of this approximation scheme is $O(n \log n + n(4/\epsilon)^{8/\epsilon^2+8/\epsilon+2})$. The computational complexity of these approximation schemes makes them hard to use in practice.

Other types of algorithms have been designed for solving the problem such as those based on enumeration techniques (Baker & Su 1974, Lageweg et al. 1976, McMahon & Florian 1975). Due to space limitation we will not review these algorithms here. Several papers such as Chen et al. (1998) and Lenstra et al. (1977) provide comprehensive surveys of scheduling problems and algorithms.

The traditional design of approximation algorithms only focused on sequential approximation algorithms for $1|r_j, q_j|C_{max}$. Here, we address an issue that was not considered in the design of approximation algorithms for this problem, that is, taking into account the huge computing power of the current multi-core systems and exploiting the potential parallelism when designing parallel approximation algorithms for $1|r_j, q_j|C_{max}$. Therefore, we design a parallel randomized approximation algorithm for $1|r_j, q_j|C_{max}$ based on the HS2 algorithm (a PTAS), proposed by Hall & Shmoys (1992). To the best of our knowledge, this is the first practical parallel approximation algorithm for $1|r_j, q_j|C_{max}$ that maintains the approximation guarantees of the sequential PTAS and is specifically designed for execution on current multi-core machines. We perform an extensive experimental analysis on a large multi-core system to evaluate the performance of the proposed algorithm using four different classes of benchmarks. The results show that for large instances of the problem, our parallel randomized algorithm achieves reasonable speedup with respect to both its sequential counterpart and the sequential EJR algorithm.

The rest of the paper is organized as follows. In Section 3, we describe the proposed parallel randomized approximation algorithm and characterize its approximation guarantee and computational complexity. In Section 4, we present the results of an extensive experimental analysis of the algorithm on a multi-core system with 64 cores. In Section 5, we conclude the paper and present possible directions for future research.

3. A parallel randomized approximation algorithm for $1|r_j, q_j|C_{max}$

We propose a parallel randomized approximation algorithm for $1|r_j, q_j|C_{max}$ that is suitable for execution on current multi-core systems and exploits their full potential for parallel execution. The proposed parallel randomized algorithm is based on the sequential PTAS (HS2) proposed by Hall & Shmoys (1992). The HS2 algorithm converts the problem into a restricted version with a fixed number of release dates by rounding the release dates. Then, it uses dynamic programming to determine the best schedule for each of the plausible choices of Δ_i's, where $\rho_i + \Delta_i$ is the time at which a job j with $r_j \geq \rho_i$ can be scheduled in the interval $[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1})$. Figure 1 illustrates the parameters of he algorithm in a schedule structure. This figure shows an instance with κ intervals. A shaded block represents the gap between the release date of each interval, ρ_i and the actual starting time of jobs with $r_j \geq \rho_i$. In this structure, a_i is the total processing time of jobs scheduled in the interval $[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1})$. Because of the large number of choices of Δ_i that need to be considered, the HS2 algorithm is not suitable for practical implementation. To provide a more practical algorithm and reduce the number of choices that need to be considered, our parallel randomized algorithm selects a sample out of all the possible choices and determines the best schedule by dynamic programming, in parallel. Thus, the basic idea of our algorithm is to avoid enumerating all possible values for Δ_i, and instead, consider a randomly selected sample of all possible choices. Table 1 gives the notation that will be used in the rest of the paper.

The proposed parallel randomized approximation algorithm for $1|r_j, q_j|C_{max}$ is given in Algorithm 1. The algorithm is designed for shared memory parallel machines with M processors. First, the algorithm rounds down all the release dates to the nearest multiple of $\frac{\epsilon}{2} \max_j\{r_j\}$ to obtain a fixed number κ of release dates, where ϵ is the approximation error (lines 1-5). Then, it employs a second phase of rounding in which the processing times and the release dates are rounded to the nearest multiple of $\frac{\epsilon P}{4n}$, where $P = \sum_{j=1}^{n} p_j$ (lines 6-9). Next, all the jobs are sorted in nonincreasing

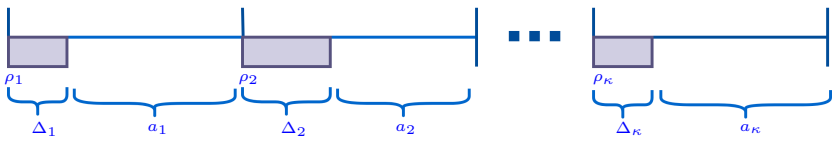


Figure 1: Illustration of the shceduling parameters used in the algorithm

Table 1: Notation

Notation	Definition
p_j	processing time of job j
r_j	release date of job j
q_j	delivery time of job j
$\widehat{P}$	sum of rounded processing times
κ	number of distinct release dates (equivalent to number of intervals)
ρ_i	release date of interval i
τ_i	actual starting time for the jobs scheduled in interval i
Δ_i	gap between ρ_i and the actual starting time τ_i for jobs with $r_j \geq \rho_i$
$c_l(\overrightarrow{a}; j)$	minimum completion time of a schedule for jobs $\{1, ..., j\}$
	that uses exactly a_i total processing time in interval $[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1})$
s	sample size
$\mathcal{D}^i$	set of possible values of Δ_i in interval i
π	set of all possible size κ vectors of Δ_i
M	number of processors (cores)
$\mathcal{C}^m$	the candidate schedule obtained by processor m

order of their delivery dates and $s = \lceil \frac{4}{\epsilon} \rceil$ random numbers are drawn from the uniform distribution within $[1, \widehat{P}]$, where $\widehat{P} = \sum_{j=1}^{n} \widehat{p}_j$. These generated numbers form the set $\mathcal{D}^i$ (lines 10-11). All the above steps are executed by only one processor.

Then the algorithm generates all possible choices of vectors of size κ out of the s elements of sets $\mathcal{D}^i$ in parallel which are then stored in the array π (lines 13-14). Once the above steps are completed the algorithm proceeds to evaluate all $(\frac{4}{\epsilon})^\kappa$ schedules in parallel, each processor out of the M processors being responsible for evaluating $(\frac{4}{\epsilon})^\kappa / M$ schedules (lines 15 - 33). For all choices of Δ_i and all possible values of a_i, jobs are scheduled such that each job is scheduled last in the interval which leads to the minimum increase in the completion time. Here, a_i is the total processing time for jobs $\{1, ..., j\}$ in the interval $[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1})$. This is done by employing dynamic programming for each choice of Δ_is, where each choice is associated with an index l. The dynamic programming formulation (Hall & Shmoys 1992) for determining the minimum completion time c_l

Algorithm 1 Parallel randomized approximation algorithm for $1|r_j, q_j|C_{max}$

$\eta \leftarrow \frac{\epsilon}{2} \max_j\{r_j\}$
for $j = 1$ **to** n **do** ▷ rounding to obtain a fixed number of release dates
    $\tilde{r}_j \leftarrow \left\lfloor \frac{r_j}{\eta} \right\rfloor$
end for
$\kappa \leftarrow$ the fixed number of release dates after rounding
$\theta \leftarrow \frac{\epsilon P}{4n}$
for $j = 1$ **to** n **do** ▷ round down all processing times and release dates
    $\widehat{p}_j \leftarrow \left\lfloor \frac{p_j}{\theta} \right\rfloor, \quad \widehat{r}_j \leftarrow \left\lfloor \frac{\tilde{r}_j}{\theta} \right\rfloor$
end for
order the jobs such that $q_1 \geq q_2 \geq ... \geq q_n$
generate $\lceil \frac{4}{\epsilon} \rceil$ random integers from $[1, \widehat{P}]$. $\mathcal{D}^i$ is the set of these generated numbers in interval i.
for $m = 1$ **to** M **do in parallel**
    pick elements $\left[\frac{\lceil \frac{4}{\epsilon} \rceil}{M}(m-1), \ldots, \frac{\lceil \frac{4}{\epsilon} \rceil}{M} m \right)$ from $\mathcal{D}^1$
    $\pi_{[1,\ldots,s^\kappa],[1,\ldots,\kappa]} \leftarrow$ all possible choices of size κ vectors formed from elements picked in line 13 and elements from $\mathcal{D}^2, \ldots, \mathcal{D}^\kappa$.
    for $l = \left[\frac{s^\kappa}{M}(m-1) \right]$ **to** $\left[\frac{s^\kappa}{M} m \right]$ **do**
        initialize the array c_l: $c_l(0, \ldots, 0; 0) \leftarrow 0$ and all other entries to ∞
        for $i = 1$ **to** κ **do**
            $\Delta_i = \pi_{l,i}$
        end for
        for $j = 1$ **to** n **do**
            for all tuples $(a_1, ..., a_\kappa)$ **satisfying**
            $(\sum_{i=1}^{\kappa} a_i = \sum_{l=1}^{j} \widehat{p}_l$ and $\rho_i + \Delta_i + a_i \leq \rho_{i+1} + \Delta_{i+1}\ \forall i = 1, ..., \kappa)$ **do**
                for all i such that $\widehat{r}_j \leq \rho_i$ **do**
                compute $c_l(a_1, ..., a_{i-1}, a_i - \widehat{p}_j, a_{i+1}, ..., a_\kappa; j)$ using equation (1)
                end for
            end for
            select the interval with the minimum length and
            schedule job j last among the jobs already scheduled in the selected interval
        end for
        keep $\min_{\overrightarrow{a}} c_l(\overrightarrow{a}; n)$ and the corresponding actual schedule
    end for
    $\mathcal{S}^m \leftarrow$ schedule corresponding to $\min_l\{\min_{\overrightarrow{a}} c_l(\overrightarrow{a}; n)\}$
end for
$\mathcal{S}_{best} \leftarrow$ schedule with minimum length among $\{\mathcal{S}^1, \mathcal{S}^2, \ldots, \mathcal{S}^M\}$
use the ordering in $\mathcal{S}_{best}$ to schedule the jobs
calculate the makespan of the obtained schedule using the original release dates and processing times

is given by:

$$c_l(\overrightarrow{a}; j) = \min\{\max\{c_l(a_1, \ldots, a_{i-1}, a_i - p_j, a_{i+1}, \ldots, a_\kappa; j-1), \rho_i + \Delta_i + a_i + q_j\}; i | r_j \leq \rho_i\} \quad (1)$$

It should be noted that to make the last interval a feasible interval, we must set the release date of interval $\kappa + 1$ to infinity, i.e., $\rho_{\kappa+1} = \infty$. The best schedule among all the schedules explored by a processor m is stored in $\mathcal{S}^m$. Then, the best schedule $\mathcal{S}_{best}$ among all schedules obtained by the M processors is determined by a parallel reduction operation with the minimum as the operator (line 34). Finally, the algorithm uses the ordering of jobs in schedule $\mathcal{S}_{best}$ to schedule the jobs and determines the makespan of the final schedule using the original release dates and processing times of the jobs. In the following, we prove that the proposed algorithm finds a schedule whose expected length is within $1 + \epsilon$ from the length of the optimal schedule.

Theorem 1. *The parallel randomized approximation algorithm for $1|r_j, q_j|C_{max}$ given in Algorithm 1 finds a schedule with an expected length of at most $(1+\epsilon)T^*$, where T^* is the length of the optimal schedule.*

Proof. In lines 6 to 9, the algorithm rounds down all the processing times and the already rounded release dates $\tilde{r}_j$ to the nearest multiple of $\theta = \epsilon P/4n$. Therefore,

$$\widehat{p}_j = \left\lfloor \frac{p_j}{\theta} \right\rfloor, \quad \widehat{r}_j = \left\lfloor \frac{\tilde{r}_j}{\theta} \right\rfloor \tag{2}$$

where $\widehat{p}_j$ and $\widehat{r}_j$ are the rounded values for p_j and r_j, respectively. Let $\widehat{P}$ be the sum of the rounded values of $\widehat{p}_j$. In line 11, the algorithm generates $\lceil \frac{4}{\epsilon} \rceil$ random numbers from the uniform distribution within $[1, \widehat{P}]$ to be assigned to $\overrightarrow{\Delta} = \{\Delta_1, \ldots, \Delta_\kappa\}$. Assume that the interval $[0, \widehat{P}]$ is divided into $\lceil \frac{4}{\epsilon} \rceil$ sub-intervals, each of length $\frac{\epsilon}{4}\widehat{P}$.

Let Δ_i^* denote the optimum value for Δ_i. Since the number of Δ_i's that are randomly generated from the uniform distribution $\mathcal{U}[1, \widehat{P}]$ is $\lceil \frac{4}{\epsilon} \rceil$, the expected value of the smallest distance between the generated Δ_i and Δ_i^* is given by,

$$E[\min_i \{|\Delta_i - \Delta_i^*|\}] = \frac{\widehat{P}}{\lceil \frac{4}{\epsilon} \rceil + 1} \leq \frac{\epsilon}{4}\widehat{P} \tag{3}$$

In the schedule with the optimum values for Δ_i, the starting time of the jobs scheduled in interval i is $\tau_i^* = \rho_i + \Delta_i^*$. With randomly generated Δ_i, the starting time is $\tau_i = \rho_i + \Delta_i$. From (3) we have,

$$\tau_i \leq \tau_i^* + \frac{\epsilon}{4}\widehat{P} \tag{4}$$

which implies that there is an increase of $\frac{\epsilon}{4}\widehat{P}$ in the makespan of the rounded problem.

The schedule in line 34 of the algorithm must be used for the problem with the original values for p_j and r_j (line 35). Thus, the increase in the makespan value will be,

$$n\theta = n\epsilon P/4n = \frac{\epsilon}{4}P \tag{5}$$

Also in line 1 of the algorithm, all the release dates are rounded to the nearest multiple of $\frac{\epsilon}{2}\max_j\{r_j\}$. Therefore, to calculate the bound for the problem with the original parameters, we just need to add $\frac{\epsilon}{2}\max_j\{r_j\}$ to every ρ_i. Clearly, there will be an increase of $\frac{\epsilon}{2}\max_j\{r_j\}$ in the makespan of the problem. We know that $\widehat{P} \leq T^*$, $P \leq T^*$, and $\max_j\{r_j\} \leq T^*$; therefore, the expected value of the makespan for the original problem is:

$$E[T] \leq T^* + \frac{\epsilon}{4}\widehat{P} + \frac{\epsilon}{4}P + \frac{\epsilon}{2}\max_j\{r_j\} \leq T^* + \frac{\epsilon}{4}T^* + \frac{\epsilon}{4}T^* + \frac{\epsilon}{2}T^* \leq T^* + \epsilon T^* \leq (1+\epsilon)T^* \tag{6}$$

where T^* is the optimal value for the makespan.

Theorem 2. *The time complexity of the proposed parallel randomized approximation algorithm for* $1|r_j, q_j|C_{max}$ *is* $O(8n(\frac{1}{\epsilon})^2(\frac{4}{\epsilon})^{\frac{2}{\epsilon}}/M)$.

Proof. The time complexity of the proposed algorithm is given by the time complexity of the parallel section (lines 12-33). Generating all possible choices of size κ vectors (lines 13-14) takes $O((\frac{1}{\epsilon})^{\frac{1}{\epsilon}}/\min\{M, \frac{1}{\epsilon}\})$.

In the next steps of the algorithm, each processor executes a loop consisting of $(\frac{4}{\epsilon})^\kappa/M$ iterations, each taking $O(n\kappa)$, for a total of $O(n\kappa(\frac{4}{\epsilon})^\kappa/M)$. Since κ is at most $\frac{2}{\epsilon}+1$, and $\frac{2}{\epsilon} > 1$, the time complexity of lines 12-33 is $O((\frac{1}{\epsilon})^{\frac{1}{\epsilon}}/\min\{M, \frac{1}{\epsilon}\} + 8n(\frac{1}{\epsilon})^2(\frac{4}{\epsilon})^{\frac{2}{\epsilon}}/M)$. For the case of $M < \frac{1}{\epsilon}$, the time complexity of the algorithm is $O((\frac{1}{\epsilon})^{\frac{1}{\epsilon}}/M + 8n(\frac{1}{\epsilon})^2(\frac{4}{\epsilon})^{\frac{2}{\epsilon}}/M)$. Therefore, the time complexity of the algorithm is given by $O(8n(\frac{1}{\epsilon})^2(\frac{4}{\epsilon})^{\frac{2}{\epsilon}}/M)$. For the case of $M > \frac{1}{\epsilon}$, the time complexity of the algorithm is $O((\frac{1}{\epsilon})^{\frac{1}{\epsilon}-1} + 8n(\frac{1}{\epsilon})^2(\frac{4}{\epsilon})^{\frac{2}{\epsilon}}/M)$. In this case, the time complexity of the algorithm is given by the second term, and therefore, it is $O(8n(\frac{1}{\epsilon})^2(\frac{4}{\epsilon})^{\frac{2}{\epsilon}}/M)$.

4. Experimental analysis

In this section, we analyze the performance of the proposed algorithm by performing extensive experiments on a multi-core system. First, we present the experimental setup, where benchmarks, distribution of the input parameters, and performance measures are defined. Then, we present the

experimental results and analyze the performance of the proposed algorithm. Hereafter, we denote the serial version of our randomized approximation algorithm by S-RAA and denote the parallel version of our randomized approximation algorithm by P-RAA.

4.1. Experimental setup

Release dates, processing times, and delivery times are the three main parameters that would affect the performance of any solution method designed for $1|r_j, q_j|C_{max}$. Among different experimental designs, analyzing the impact of the variance of the release date and the ratio of the delivery time to the processing time is of high importance. Therefore, to provide insights on the performance of our proposed algorithm on different configurations of these parameters, we employ several benchmark problem instances classified into four classes, as follows:

I. *Short delivery times, high variance release dates* (*SD-HVR*): the delivery times are not very large compared to the processing times, and the release dates have a high variance.

II. *Long delivery times, high variance release dates* (*LD-HVR*): the delivery times are large compared to the processing times, and the release dates have a high variance.

III. *Short delivery times, low variance release dates* (*SD-LVR*): the delivery times are not very large compared to the processing times, and the release dates have a low variance.

IV. *Long delivery times, low variance release dates* (*LD-LVR*): the delivery times are large compared to the processing times, and the release dates have a low variance.

In this analysis, we rely on independently generated random instances. The size of each instance is determined by the number of jobs, n. Also, there are three input parameters for each instance: (i) processing times, p_j, which are drawn from a normal distribution; (ii) release dates, r_j, which are drawn from a uniform distribution; and (iii) delivery times, q_j, which are drawn from a normal distribution. In the experiments, we investigate the impact of the magnitude of delivery dates with respect to processing times on the performance of our proposed algorithm. For that reason, we take processing times as the base with a fixed distribution across four class of instances and change the parameters for the distribution of delivery dates. The distributions of the input parameters for each class of instances are shown in Table 2, where $i \in \{3, \ldots, 7\}$. In this table, we denote by $U[x, y]$, the uniform distribution within interval $[x, y]$, and by $N(\mu, \sigma)$, the normal distribution with mean μ and standard deviation σ.

Table 2: Distribution of parameters for each class of benchmarks

Class	p_j	r_j	q_j
SD-HVR	$N(100, 20)$	$U(0, 10^i)$	$N(100, 20)$
LD-HVR	$N(100, 20)$	$U(0, 10^i)$	$N(500, 100)$
SD-LVR	$N(100, 20)$	$U(0, 10^2)$	$N(100, 20)$
LD-LVR	$N(100, 20)$	$U(0, 10^2)$	$N(500, 100)$

We analyze the performance of the proposed algorithm for each class of instances using two important metrics, the execution time and the makespan obtained by the proposed parallel randomized approximation algorithm, for instances of various sizes. For benchmarking, we can not rely on the existing approximation schemes (HS2 and HS3) due to their very high time complexity, which makes them unfeasible to execute in reasonable amount of time. Therefore, we use a constant approximation algorithm as a benchmark. There exist two approximation algorithms with the lowest time complexity, i.e., Extended Jackson's Rule (EJR), proposed by Schrage (1971), and the approximation algorithm proposed by Nowicki & Smutnicki (1994). Although these two algorithms have the same asymptotic time complexity, we choose EJR for benchmarking since this algorithm is the building block of the algorithm by Nowicki & Smutnicki (1994), and is expected to have lower actual running time than it.

We define the *performance gap with respect to* EJR as $G_{EJR} = \frac{C - C^{EJR}}{C^{EJR}}$, where C is the makespan of the schedule obtained by the proposed approximation algorithm, and C^{EJR} is the makespan obtained by the EJR algorithm. For the analysis of the execution time, we compare the execution time of the parallel approximation algorithm with that of EJR which has the time complexity of $O(n \log n)$. This analysis is performed by varying the number of cores from 2 to 64 and with instances of sizes $10^i,\ i \in \{3, \ldots, 7\}$. Since we perform our analysis using $\epsilon \geq 0.2$, we do not expect considerable speed-up by executing lines 13-14 of the algorithm on multiple cores. Thus, we use one core for this part of the algorithm. To analyze the effects of different values of ϵ on the execution time and the speed-up of the proposed algorithm, we perform the experiments with $\epsilon = \{0.2, 0.6, 1.0\}$. The aim of including $\epsilon = 1.0$ in our analysis is to maintain fairness while comparing the performance of our proposed algorithm with that of EJR. We define two speed-up metrics to evaluate the performance of the parallel approximation algorithm: (i) *speed-up with respect to EJR*, denoted by S_{EJR}, and (ii) *speed-up with respect to the serial version of the*

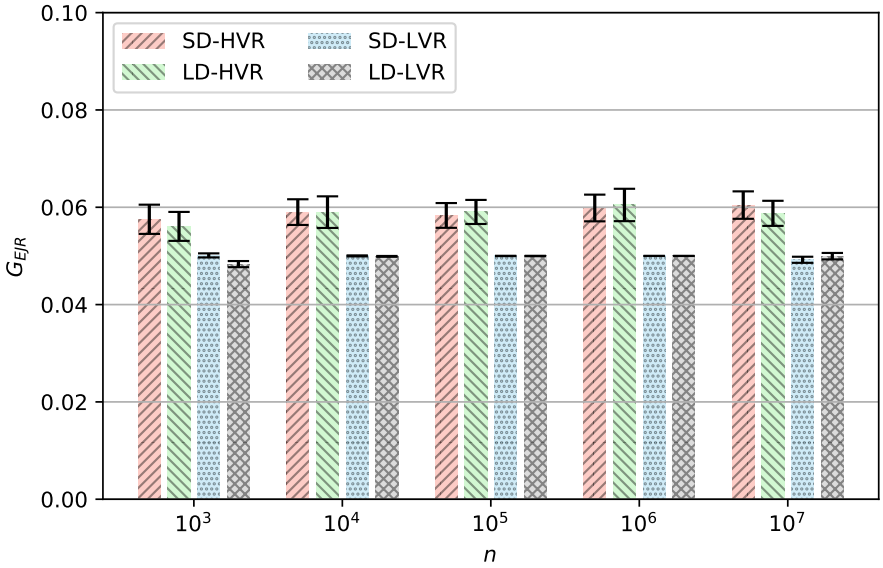
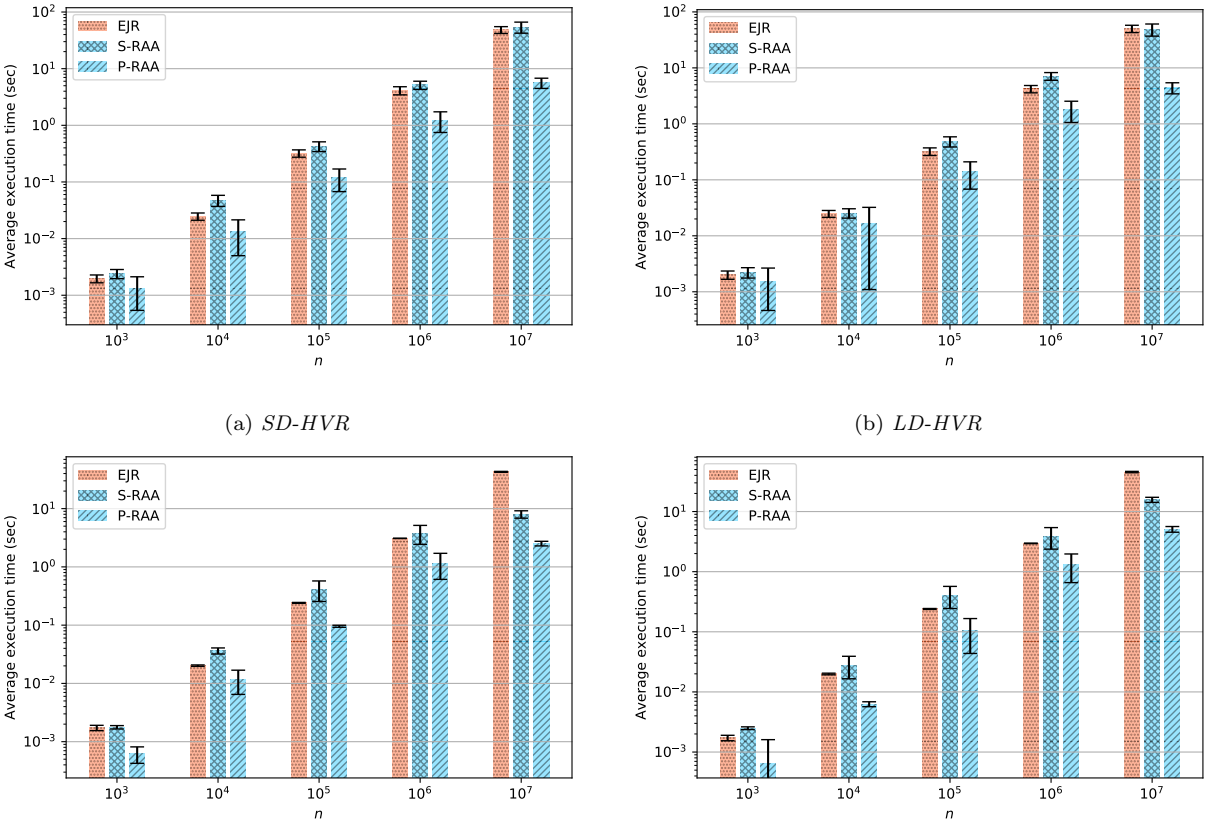


Figure 2: Performance gap with respect to EJR vs. size of instances ($\epsilon = 1.0$)

proposed parallel approximation algorithm, denoted by S_S. These metrics are defined as follows: $S_{EJR} = \frac{T_{EJR}}{T_P}$ and $S_S = \frac{T_S}{T_P}$, where T_{EJR} is the average execution time of EJR, T_P is the average execution time of the proposed parallel randomized approximation algorithm, and T_S is the average execution time of the serial version of the proposed parallel randomized approximation algorithm. All the experiments are performed on a multi-core AMD system with 64-cores, 2.4GHz, and 512GB of RAM. Each experiment is run 30 times and the analysis is done based on the average values of the makespan, and the running time. We use standard deviation (STD) as a metric for the reliability of the results (McGeoch 1992, Johnson 2002). Also, in order to provide some on the range of the execution time and G_{EJR} for the considered instances, we use the error bar histograms showing the standard deviation of the metrics obtained by considering the 30 runs.

4.2. Experimental results

In this section, we present the experimental results and aim at providing insights on the performance of the proposed randomized approximation algorithm under different distributions of the parameters. First in Figure 2, we compare the gap in performance of S-RAA with respect to EJR for different classes of benchmarks. We observe that the gap (G_{EJR}) is within 6% for all classes of instances, which is acceptable due to the fact that EJR is a 2-approximation algorithm and S-RAA



(a) *SD-HVR*

(b) *LD-HVR*

(c) SD-LVR

(d) LD-LVR

Figure 3: Execution time vs. size of instances

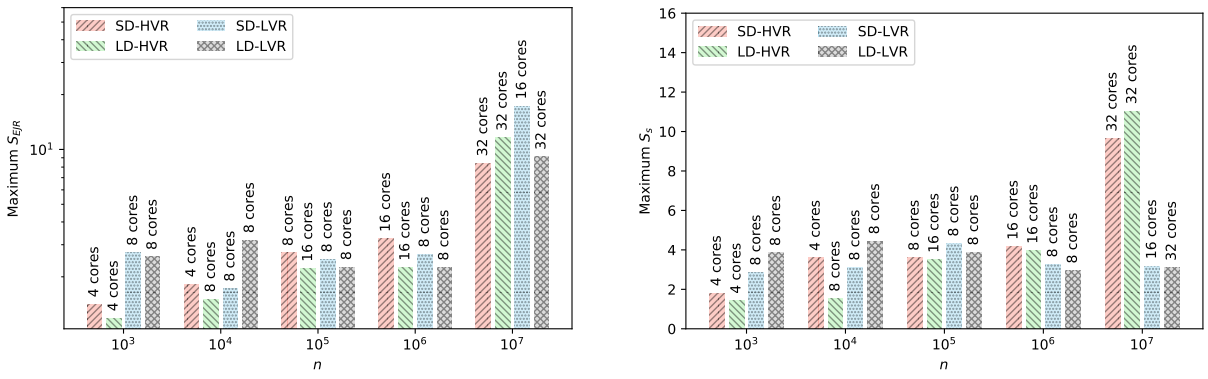
has the same approximation ratio with $\epsilon = 1.0$. We also observe that generally, S-RAA performs better for the classes of benchmarks with low variance for the release dates, i.e., *SD-LVR* and *LD-LVR*. Comparing the performance of on instances with short delivery times and long delivery times shows that our algorithm is not significantly sensitive to the relative distribution of the processing and delivery times.

Now, we evaluate the performance of S-RAA and P-RAA algorithms with respect to EJR. Since EJR is a 2-approximation algorithm, for the sake of fairness, we run S-RAA and P-RAA with $\epsilon = 1.0$. We solve instances of sizes 10^i, $i \in \{3, \ldots, 7\}$ using EJR, S-RAA, and P-RAA, and analyze the average execution time for each size. For P-RAA, we use the minimum average execution time across different number of cores. Figure 3 shows the execution times for the four class of instances.

In Figure 3a, the results for class *SD-HVR* are shown for instances with different sizes. This class of benchmarks includes instances in which the delivery times of jobs are not very large compared to the processing times. Another feature of this class of benchmarks is that the distribution of the arrival time of jobs (release date) has a high variance. In this figure, we observe that S-RAA has a slightly higher execution time compared to the EJR algorithm. But, by using the parallel algorithm (P-RAA) and using multiple cores, we obtain a smaller execution time for all the sizes of instances. P-RAA obtains the minimum average execution times with 4 cores for instances of sizes 10^3 and 10^4. P-RAA requires 8, 16, and 32 cores to acheive the minimum average execution times for instances of sizes 10^5, 10^6, and 10^7 respectively. It is clear that as the size of instances increases, a larger number of cores is needed to obtain the best execution times. In other words, jobs in a larger size instance have a larger number of fixed release dates resulting in a larger number of intervals and a bigger set of choices for Δ_i's. Therefore, for large instances P-RAA requires more cores to achieve the highest possible speed-up ratios.

Figure 3b shows the results for class *LD-HVR* for instances of different sizes.. This class of benchmarks consists of instances in which the delivery times of the jobs are larger than their processing times and the release dates have a high variance. We observe that the results for this class of benchmarks resemble those obtained in the case of the *SD-HVR* class of benchmarks. For this class of benchmarks, the EJR algorithm outperforms the S-RAA. But the execution time of P-RAA is significantly smaller than that of EJR. The lowest execution time of P-RAA is obtained with 4 cores for instances of size 10^3, 8 cores for instances of size 10^4, 16 cores for instances of size 10^5 and 10^6, and 32 cores for instances of size 10^7.

Figure 3c shows the results for class *SD-LVR* for instances of different sizes. This class of benchmarks, includes instances in which the delivery times of jobs are comparable to the their processing times, and the distribution of the arrival times of jobs (release dates) has a low variance. The results of our analysis show a significant reduction in the execution time of S-RAA and P-RAA compared to the first two classes of benchmarks (*SD-HVR* and *LD-HVR*), specially for the large size of instances. For the *SD-LVR* class, similar to the first two classes of benchmarks, S-RAA and P-RAA are more competitive for large size instances. We observe that for instances of size 10^7, S-RAA obtains the solution within 10 seconds and outperforms EJR. We observe that P-RAA requires 8 cores to obtain the minimum execution times for the instances of size smaller than 10^7.



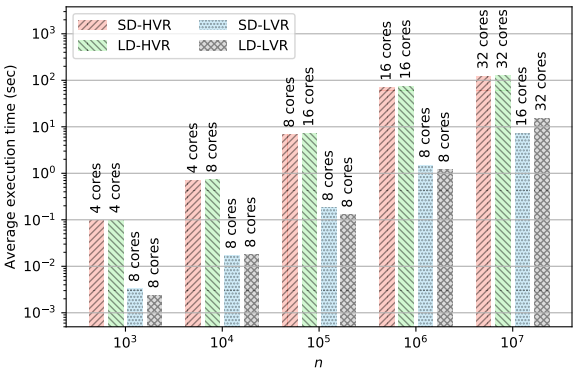
(a) S_{EJR} (b) S_s

Figure 4: Maximum speed-ups vs. size of instances ($\epsilon = 1.0$)

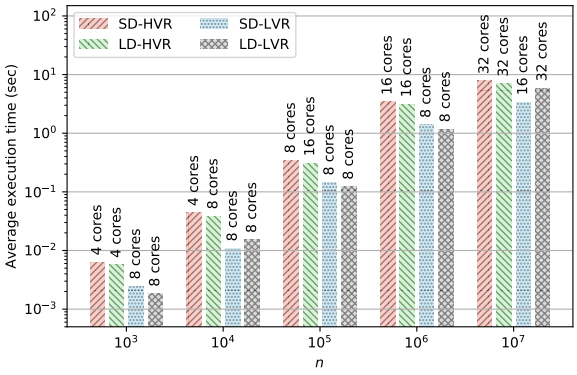
For instances of size 10^7, the best execution time is obtained with 16 cores.

Figure 3d shows the results for class *LD-LVR* which consists of benchmarks with jobs with low variance release dates, and the processing times of jobs are small compared to their delivery times. The execution time of EJR for this class of benchmarks is almost within the same range as in the case of *SD-LVR* instances. On the other hand, the execution time of S-RAA for large size instances is slightly higher than its execution time for the *SD-LVR* class, indicating that our algorithm is more sensitive to the relative distribution of the processing and delivery times than the EJR algorithm. We also observe that for the class of instances of size $n \leq 10^6$, EJR outperforms S-RAA in terms of the execution time. But for larger size instances, the execution time of S-RAA is significantly smaller than that of EJR. Similar to the other classes of benchmarks, as the size of instances increases, the difference between the execution time of P-RAA and the execution time of EJR becomes more significant. For the class of instances of size $n \leq 10^6$, the best execution times for P-RAA are obtained with eight cores, while the best execution time for the instances of size 10^7 is obtained with 32 cores.

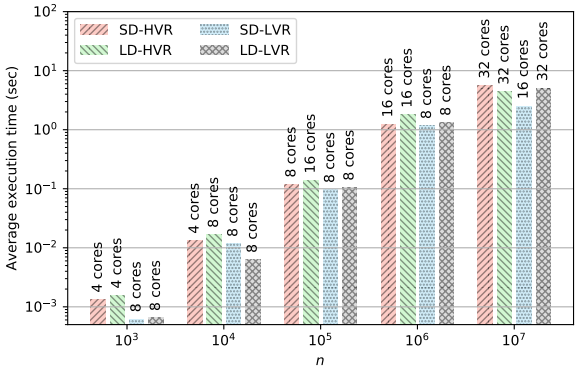
Figure 4 shows the maximum speed-up ratios for different size of instances from different classes of benchmarks. Figure 4a shows the maximum speed-up ratio with respect to EJR, S_{EJR}. The number of required cores for obtaining the maximum speedup is also provided. We observe an increasing trend in the maximum S_{EJR}, which indicates that P-RAA performs better than EJR in the case of large size instances. For small size instances, the relative magnitude of the parallelization



(a) $\epsilon = 0.2$



(b) $\epsilon = 0.6$



(c) $\epsilon = 1.0$

Figure 5: Best execution time vs. size of instances

overhead prevents the algorithm to yield a significant speed-up. No significant difference is observed in the maximum speed-up ratio, S_{EJR}, among different classes of benchmarks.

One important metric in the evaluation of P-RAA is the speed-up with respect to its serial version (S_s). Figure 4b shows the maximum speed-up ratio of P-RAA with respect to S-RAA. We do not observe any significant difference in the maximum S_s for instances of size $n \leq 10^6$. We observe a significantly higher S_s for instances of size 10^7 for classes *SD-HVR* and *LD-HVR*. The reason is that the instances within these two classes have a larger expected number of intervals, and therefore, more permutations for the values of Δ_i. In this case, there is more space for P-RAA to gain speed-up with respect to S-RAA.

We also compare the best execution times of P-RAA for instances of different sizes (10^i, $i \in \{3, \ldots, 7\}$) from the four classes of benchmarks (Figure 5). We observe that the execution times of

the proposed algorithm for the *SD-HVR* and *LD-HVR* classes are much higher than the execution times obtained for the other two classes of benchmarks (i.e., *SD-LVR* and *LD-LVR*). This indicates that the execution time of the proposed parallel randomized approximation algorithm is highly correlated with the variance of the release dates. This trend is observed for all three different values of ϵ considered here. The difference between the execution time in classes with high variance release dates and low variance release dates is more significant for small values of ϵ. On the other hand, no significant difference is observed in the performance of the proposed algorithm between *SD-HVR* and *LD-HVR* classes, and also between *SD-LVR* and *LD-LVR* classes. This indicates that the execution time of the proposed algorithm is not very sensitive to the relative distribution of the processing and delivery times.

In summary, the experimental analysis shows that our proposed algorithm can solve very large-scale single machine scheduling problems within few seconds which makes it suitable for a wide range of real world applications where the scheduler must obtain a quality solution in real time. Task scheduling in parallel computing (Blumofe & Park 1994), scheduling of large-scale data broadcasting (Aksoy & Franklin 1998, Pozo et al. 2019), and task scheduling in big data and High Performance Computing (HPC) (Reuther et al. 2018, Rjoub et al. 2020, Fu et al. 2019) are few examples of the areas where our proposed algorithm can be employed efficiently. It is also important to note that the proposed algorithm is an approximation scheme that provides users with the flexibility of balancing between the running time of the algorithm and the quality of solutions.

5. Conclusion

We designed a parallel randomized approximation algorithm for solving the non-preemptive single machine scheduling problem with release dates and delivery times, $1|r_j, q_j|C_{max}$. We performed an extensive experimental analysis to evaluate the performance of the proposed algorithm. The experimental results show that the solutions obtained by the proposed parallel approximation algorithm are within a small gap of those obtained by the Extended Jackson's Rule (EJR) algorithm. They also show that the proposed algorithm obtains very good speedup with respect to its own sequential version, as well as with respect to EJR, which has the time complexity of (i.e., $O(n \log n)$). The $1|r_j, q_j|C_{max}$ problem is widely used as a sub-problem in several algorithms for shop scheduling problems, therefore, as a future research, we plan to employ the proposed approximation algorithm

in solving the sub-problems of various shop scheduling problems. We also plan to design parallel approximation algorithms for other NP-hard combinatorial optimization problems.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, *34*, 391–401.

Aksoy, D., & Franklin, M. (1998). Scheduling for large-scale on-demand data broadcasting. In *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98* (pp. 651–659). IEEE volume 2.

Allahverdi, A., Ng, C., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*, 985–1032.

Augusto, V., Xie, X., & Perdomo, V. (2010). Operating theatre scheduling with patient recovery in both operating rooms and recovery beds. *Computers & Industrial Engineering*, *58*, 231–238.

Baker, K. R., & Su, Z.-S. (1974). Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Research Logistics Quarterly*, *21*, 171–176.

Balas, E., Lenstra, J. K., & Vazacopoulos, A. (1995). The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, *41*, 94–109.

Bittencourt, L. F., Diaz-Montes, J., Buyya, R., Rana, O. F., & Parashar, M. (2017). Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, *4*, 26–35.

Blelloch, G. E., Peng, R., & Tangwongsan, K. (2011). Linear-work greedy parallel approximate set cover and variants. In *Proc. of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures* (pp. 23–32).

Blelloch, G. E., & Tangwongsan, K. (2010). Parallel approximation algorithms for facility-location problems. In *Proc. of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures* (pp. 315–324).

Blumofe, R. D., & Park, D. S. (1994). Scheduling large-scale parallel computations on networks of workstations. In *Proceedings of 3rd IEEE International Symposium on High Performance Distributed Computing* (pp. 96–105). IEEE.

Cardoen, B., Demeulemeester, E., & Beliën, J. (2010). Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, *201*, 921–932.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, *11*, 42–47.

Carlier, J., & Pinson, É. (1989). An algorithm for solving the job-shop problem. *Management Science*, *35*, 164–176.

Chen, B., Potts, C. N., & Woeginger, G. J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of combinatorial optimization* (pp. 1493–1641). Springer.

Dulebenets, M. A. (2019). A delayed start parallel evolutionary algorithm for just-in-time truck scheduling at a cross-docking facility. *International Journal of Production Economics*, *212*, 236–258.

Fazlirad, A., & Brennan, R. W. (2018). Multiagent manufacturing scheduling: An updated state of the art review. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)* (pp. 722–729). IEEE.

Feng, Z.-K., Niu, W.-J., Zhou, J.-Z., Cheng, C.-T., Qin, H., & Jiang, Z.-Q. (2017). Parallel multi-objective genetic algorithm for short-term economic environmental hydrothermal scheduling. *Energies*, *10*, 163.

Fu, W., Liu, S., & Srivastava, G. (2019). Optimization of big data scheduling in social networks. *Entropy*, *21*, 902.

Funabiki, N., & Takefuji, Y. (1993). A parallel algorithm for broadcast scheduling problems in packet radio networks. *IEEE Transactions on Communications*, *41*, 828–831.

Gahm, C., Kanet, J. J., & Tuma, A. (2019). On the flexibility of a decision theory-based heuristic for single machine scheduling. *Computers & Operations Research*, *101*, 103 – 115.

Garey, M. R., & Johnson, D. S. (1978). "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)*, *25*, 499–508.

Ghalami, L., & Grosu, D. (2017). A parallel approximation algorithm for scheduling parallel identical machines. In *Proc. of the 7th IEEE Workshop on Parallel / Distributed Computing and Optimization (PDCO 2017)* (pp. 442–451).

Ghalami, L., & Grosu, D. (2019). Scheduling parallel identical machines to minimize makespan: A parallel approximation algorithm. *Journal of Parallel and Distributed Computing*, *133*, 221–231.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics* (pp. 287–326). Elsevier volume 5.

Gupta, D., & Denton, B. (2008). Appointment scheduling in health care: Challenges and opportunities. *IIE Transactions*, *40*, 800–819.

Hall, L. A., & Shmoys, D. B. (1992). Jackson's rule for single-machine scheduling: Making a good heuristic better. *Mathematics of Operations Research*, *17*, 22–35.

Hall, R. W. et al. (2012). *Handbook of healthcare system scheduling*. Springer.

Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, *59*, 215–250.

Kalashnikov, A., & Kostenko, V. (2008). A parallel algorithm of simulated annealing for multiprocessor scheduling. *Journal of Computer and Systems Sciences International*, *47*, 455–463.

Kise, H., Ibaraki, T., & Mine, H. (1979). 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*, *22*, 205–224.

Kurz, M., & Askin, R. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, *39*, 3747–3769.

Laalaoui, Y., & M'Hallah, R. (2016). A binary multiple knapsack model for single machine scheduling with machine unavailability. *Computers & Operations Research*, *72*, 71 – 82.

Lageweg, B., Lenstra, J. K., & Kan, A. (1976). Minimizing maximum lateness on one machine: Computational experience and some applications. *Statistica Neerlandica*, *30*, 25–41.

Lawler, E. L., Lenstra, J. K., Kan, A. H. R., & Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, *4*, 445–522.

Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, *1*, 343–362.

Li, Y., Ghalami, L., Schwiebert, L., & Grosu, D. (2018). A gpu parallel approximation algorithm for scheduling parallel identical machines to minimize makespan. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (pp. 619–628). IEEE.

Marynissen, J., & Demeulemeester, E. (2019). Literature review on multi-appointment scheduling problems in hospitals. *European Journal of Operational Research*, *272*, 407–419.

Mashayekhy, L., Nejad, M. M., Grosu, D., Zhang, Q., & Shi, W. (2015). Energy-aware scheduling of mapreduce jobs for big data applications. *IEEE Transactions on Parallel and Distributed Systems*, *26*, 2720–2733.

McGeoch, C. (1992). Analyzing algorithms by simulation: variance reduction techniques and simulation speedups. *ACM Computing Surveys (CSUR)*, *24*, 195–212.

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

Mnich, M., & van Bevern, R. (2018). Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, *100*, 254–261.

Nguyen, S., Mei, Y., & Zhang, M. (2017). Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, *3*, 41–66.

Niu, S., Song, S., Ding, J.-Y., Zhang, Y., & Chiong, R. (2019). Distributionally robust single machine scheduling with the total tardiness criterion. *Computers & Operations Research*, *101*, 13 – 28.

Nowicki, E., & Smutnicki, C. (1994). An approximation algorithm for a single-machine scheduling problem with release times and delivery times. *Discrete Applied Mathematics*, *48*, 69–79.

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

Pozo, F., Rodriguez-Navas, G., & Hansson, H. (2019). Methods for large-scale time-triggered network scheduling. *Electronics*, *8*, 738.

Rajagopalan, S., & Vazirani, V. V. (1998). Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM J. Comput.*, *28*, 525–540.

Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A. et al. (2018). Scalable system scheduling for hpc and big data. *Journal of Parallel and Distributed Computing*, *111*, 76–92.

Rjoub, G., Bentahar, J., & Wahab, O. A. (2020). Bigtrustscheduling: Trust-aware big data task scheduling approach in cloud computing environments. *Future Generation Computer Systems*, *110*, 1079–1097.

Schoenfelder, J., Bretthauer, K. M., Wright, P. D., & Coe, E. (2020). Nurse scheduling with quick-response methods: Improving hospital performance, nurse workload, and patient experience. *European Journal of Operational Research*, *283*, 390–403.

Schrage, L. (1971). Obtaining optimal solutions to resource constrained network scheduling problems. *Unpublished manuscript*, *189*.

Shen, J., & Zhu, Y. (2020). A single machine scheduling with periodic maintenance and uncertain processing time. *International Journal of Computational Intelligence Systems*, *13*, 193–200.

Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, *6*, 108–117.

Talebi, J., Badri, H., Ghaderi, F., & Khosravian, E. (2009). An efficient scatter search algorithm for minimizing earliness and tardiness penalties in a single-machine scheduling problem with a common due date. In *2009 IEEE Congress on Evolutionary Computation* (pp. 1012–1018).

Tsai, C.-W., Huang, W.-C., Chiang, M.-H., Chiang, M.-C., & Yang, C.-S. (2014). A hyper-heuristic scheduling algorithm for cloud. *IEEE Transactions on Cloud Computing*, *2*, 236–250.

Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.

Weng, C., & Lu, X. (2005). Heuristic scheduling for bag-of-tasks applications in combination with qos in the computational grid. *Future Generation Computer Systems*, *21*, 271–280.

Williamson, D. P., & Shmoys, D. B. (2011). *The Design of Approximation Algorithms*. Cambridge University Press.

Zhao, Z., Liu, S., Zhou, M., Guo, X., & Qi, L. (2020). Decomposition method for new single-machine scheduling problems from steel production systems. *IEEE Transactions on Automation Science and Engineering*, *17*, 1376–1387.