Extended Abstract

Master Thesis

A Parallel Randomized Approximation Algorithm For Single Machine Scheduling With Applications To Flow Shop Scheduling

by

HOSSEIN BADRI
Department of Computer Science, Wayne State University, Detroit, MI 48202, USA
Email: *hossein.badri@wayne.edu*

1 Introduction

Scheduling has received a significant attention by scholars in the field of operations research and design of algorithms, resulting in a large number of publications in the literature. There are two main reasons for such significant efforts on developing novel algorithms for scheduling problems. First, scheduling has extensive applications in several domains. Scheduling problems have extensively been applied in different problems in manufacturing systems, healthcare systems, distributed systems, etc. Second, most of the scheduling problems are known to be computationally complex, that is, commercial solvers are not able to solve real size problem instances in reasonable amount of time. 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. This feature has motivated scholars to design efficient algorithms which are capable of solving scheduling problems, efficiently. Heuristic methods have been among the most widely used methods for solving scheduling problems. Heuristic methods are usually efficient in terms of the execution time, while the main drawback of these methods is the lack of any guarantees for the quality of solutions. Given this fact, a significant amount of efforts in the area of scheduling has been 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 (so-called the approximation ratio or performance guarantee of the algorithm). In addition to the significant work on developing algorithms for scheduling problems, some scholars devoted their efforts on leveraging the huge amount of computing power of the current multicore computing systems and develop efficient parallel algorithms. Though, considerable attention has not been devoted to the development of parallel approximation algorithms and the majority of works have been focused on parallel heuristic algorithms.

In this thesis, we address the single machine scheduling which 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. We design a parallel randomized approximation algorithm for the non-preemptive single machine scheduling problem with release dates and delivery times, where the objective is to minimize the completion time of all jobs (i.e., makespan). This problem is denoted by $1|r_j, q_j|C_{max}$, where r_j is the release date and q_j is the delivery time of job j, and C_{max} is the completion time of all jobs. We employ the proposed approximation algorithm to design efficient parallel algorithms for flow shop scheduling problem denoted by $F \mid\mid C_{max}$, where F specifies that the environment is a flow shop, and C_{max} specifies the objective, that is, the minimization of the makespan. The flow shop scheduling problem is one of the most complex scheduling problems, and finding an optimal solution for real size instances in a reasonable amount of time is practically infeasible. In this thesis, our aim is to leverage the huge amount of computing power of the current multicore computing systems and develop efficient parallel algorithms to solve large size flow shop scheduling problems in reasonable amount of time. We design two parallel algorithms based on the Shifting Bottleneck (SB) heuristic. The first one is a coarse-grained parallel algorithm that is suitable for execution on multi-core systems with a small number of cores, while the second one is a fine-grained parallel algorithm suitable for multi-core systems with a large number of cores. We perform extensive experimental analyses to evaluate the performance of the proposed algorithms for instances of various sizes.

1.1 Contributions of this research

Most variants of the single machine scheduling problems are known to be NP-hard, and therefore, many efforts have been devoted to the development of approximation algorithms for solving them. In this thesis, 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). The majority of research in approximation algorithms has been focused on designing sequential approximation algorithms with various approximation guarantees for a large variety of NP-hard combinatorial optimization problems [14]. Very few efforts were directed at designing parallel approximation algorithms for such problems. To the best of our knowledge, we are not aware of any research addressing the design of parallel approximation algorithms for the problem of non-preemptive single machine scheduling with release dates and delivery times to minimize the makespan.

In this thesis, we also address the flow shop scheduling problem as an application of the proposed algorithm for single machine scheduling problem. We design two Shifting Bottleneck-based parallel algorithms for the flow shop scheduling problem. The first one is a *coarse-grained parallel algorithm* that is suitable for execution on multi-core systems with a small number of cores, while the second one is a *fine-grained parallel algorithm* suitable for multi-core systems with a large number of cores. The results of our experimental analysis on the proposed parallel algorithms for the flow shop scheduling problem show that the proposed algorithms can solve large-size instances of the problem in a reasonable amount of time and obtain solutions that are within acceptable distance from the lower bounds. The proposed parallel algorithms achieve good speedup with respect to the serial variants of the algorithms.

The research presented in this dissertation has been submitted to top journals in operations research including *Computers & Operations Research* [2] and *IEEE Transactions on Systems, Man and Cybernetics: Systems* [3].

2 A Parallel Randomized Approximation Algorithm for Non-Preemptive Single Machine Scheduling

Single machine scheduling problems have received a significant attention from researchers in the field of operations research and algorithms, resulting in a large number of publications. The majority of research in approximation algorithms has been focused on designing sequential approximation algorithms with various approximation guarantees for a large variety of NP-hard combinatorial optimization problems. Very few efforts were directed at designing parallel approximation algorithms for such problems. To the best of our knowledge, we are not aware of any research addressing the design of parallel approximation algorithms for the problem of non-preemptive single machine scheduling with release dates and delivery times to minimize the makespan. This problem is denoted by $1|r_j, q_j|C_{max}$ 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

Table 1: Sequential approximation algorithms for $1|r_j, q_j|C_{max}$

Algorithm	Approximation ratio	Complexity
Schrage (EJR) [13]	2	$O(n \log n)$
Potts [12]	3/2	$O(n^2 \log n)$
Hall and Shmoys (HS1) [6]	4/3	$O(n^2 \log n)$
Nowicki and Smutnicki [11]	3/2	$O(n \log n)$
Hall and Shmoys (HS2) [6]	$1+\epsilon$	$O(16^{1/\epsilon}(n/\epsilon)^{3+4/\epsilon})$
Hall and Shmoys (HS3) [6]	$1+\epsilon$	$O(n \log n + n(4/\epsilon)^{8/\epsilon^2+8/\epsilon+2}$

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 ($1|r_j|L_{max}$). Algorithms for solving this problem have been widely employed as building blocks in the design of algorithms for shop scheduling problems. This problem is strongly NP-hard [10], and therefore, a polynomial time approximation scheme (PTAS) is the best algorithm one can hope to design for it, unless $P = NP$. A PTAS is a family of $(1 + \epsilon)$-approximation algorithms whose time complexity can depend arbitrarily on $1/\epsilon$. Our goal in this research 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. The traditional design of approximation algorithms only focused on sequential approximation algorithms for $1|r_j, q_j|C_{max}$. Table 1 gives a review of existing sequential approximation algorithms for $1|r_j, q_j|C_{max}$, their approximation ratios, and their complexity. Other types of algorithms have been designed for solving the problem such as those based on enumeration techniques.

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 and Shmoys [6]. 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.

2.1 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 and Shmoys [6]. 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 Δ_i is such that $\rho_i + \Delta_i$ is the time at which a job j with $r_j \geq \rho_i$ can be scheduled in the interval $[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1})$. Because of the large number of choices 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. Table 2 gives the notation that will be used in the rest of the chapter.

Table 2: Notations

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 the interval $[\rho_i+\Delta_i, \rho_{i+1}+\Delta_{i+1})$
s	sample size
$\mathcal{D}$	set of possible values of Δ_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

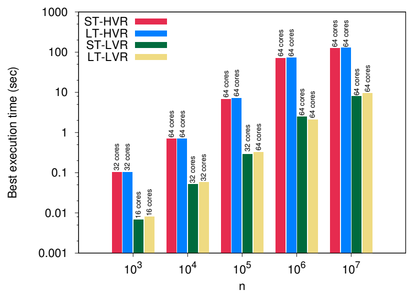
The algorithm is designed for shared memory parallel machines with M processors. First, the algorithm rounds down all the release dates to the nearest multiple of $\frac{\epsilon}{2}\max_j\{r_j\}$ to obtain a fixed number κ of release dates, where ϵ is the approximation error. Then, it employs a second 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 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 $\mathcal{D}$. The algorithm generates all possible choices of vectors of size κ out of the s elements of set $\mathcal{D}$ which are then stored in the array π. All the above steps are executed by only one processor. 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. 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 [6] for determining the minimum completion time c_l is given by:

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

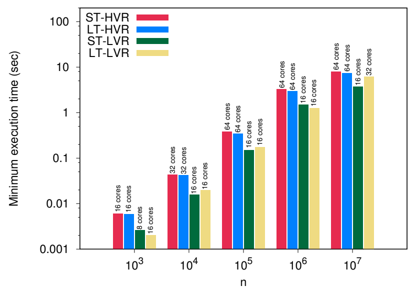
It should be noted that to make the last interval a feasible interval, we must set the release date of interval $\kappa+1$ to infinity, i.e., $\rho_{\kappa+1}=\infty$. The best schedule among all the schedules explored by a processor m is stored in $\mathcal{S}^m$. Then, the best schedule $\mathcal{S}_{best}$ among all schedules obtained by the M processors is determined by a parallel reduction operation with the minimum as the operator. 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 proposed parallel randomized approximation algorithm for $1|r_j,q_j|C_{max}$ finds a schedule with an expected length of $(1+\epsilon)T^*$, where T^* is the length of the optimal schedule.*

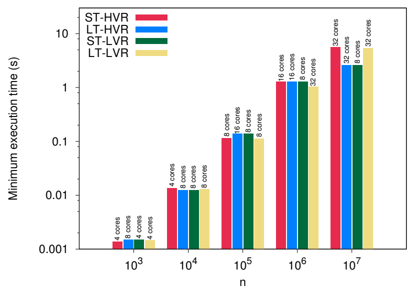
Theorem 2 *The time complexity of the proposed parallel randomized approximation algorithm for $1|r_j,q_j|C_{max}$ is $O(8n(\frac{1}{\epsilon})^2(\frac{4}{\epsilon})^{\frac{2}{\epsilon}}/M)$.*



(a) $\epsilon = 0.2$



(b) $\epsilon = 0.6$



(c) $\epsilon = 1.0$

Figure 1: Best execution time vs. size of instances

2.2 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. 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 some insights about 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 time, 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 time, 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 time, 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 time, 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.

We present a summary of our experimental analysis by comparing the results for the four classes of benchmarks. We aim at providing insights on the performance of the proposed parallel randomized approximation algorithm under different distributions of the parameters. First, we compare the gap in performance of the proposed randomized approximation algorithm with respect to *EJR* for different classes of benchmarks. We observe that generally, the proposed algorithm performs better for the classes of benchmarks with low variance for the release dates, i.e., *SD-LVR* and *LD-LVR*. Comparing short delivery time and long delivery time classes shows that the performance of our algorithm is not significantly sensitive to the relative distribution of the processing and delivery times.

We also compare the best execution times of the algorithms for different sizes of instances (10^i, $i \in \{3, \ldots, 7\}$) under four classes of benchmarks (Figure 1). We observe that the execution times of the

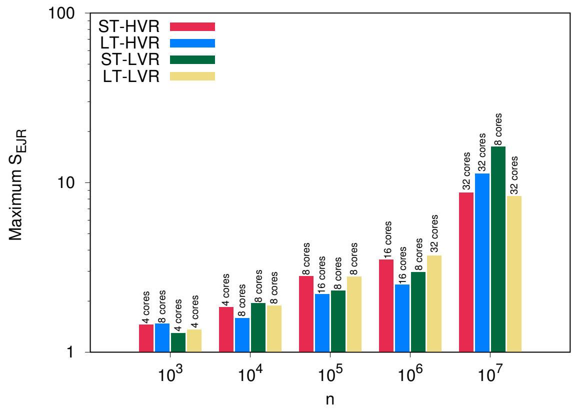


Figure 2: Maximum speed-up with respect to *EJR* vs. size of instances ($\epsilon = 1.0$)

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 the 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 class *SD-HVR* and *LD-HVR*, and also between class *SD-LVR* and *LD-LVR*. This indicates that the execution time of the proposed algorithm is not very sensitive to the relative distribution of the processing and delivery times.

Figure 2 shows the maximum speed-up ratio with respect to *EJR*, S_{EJR}, for instances of various sizes from different classes of benchmarks. 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 the parallel approximation algorithm performs better than *EJR* in the case of large size instances. For small size instances, the relative magnitude of the parallelization 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.

3 Parallel Shifting Bottleneck Algorithms for Flow Shop Scheduling

Among all the scheduling problems, the flow shop scheduling problem is one of the most complex and widely applicable. In this research, we focus on the general flow-shop scheduling problem which has as objective the minimization of the makespan. Using the notation for scheduling problems proposed by Lawler et al. [9], the flow shop scheduling problem is denoted by $F \mid\mid C_{max}$, where F specifies that the environment is a flow shop, and C_{max} specifies the objective, that is, the minimization of the makespan. The $F \mid\mid C_{max}$ problem is defined as follows:

> $F \mid\mid C_{max}$ *Problem*: We are given a set of n jobs that need to be processed on a set of m machines located in a given order. Each job j, $j = 1, \ldots, n$, consists of a sequence of m operations, one operation for each of the m machines. Operation i of job j executed on machine i requires t_{ij} units of time. All jobs must be processed in the given order of the machines and preemption is not allowed. Each machine can process at most one operation at a time and each operation of a job starts processing only after all the preceding operations of the job have been completed. The objective is to find a feasible schedule that *minimizes the makespan* C_{max}, where $C_{max} := \max_{j=1,\ldots,n} C_j$, and C_j is the completion time of the last

operation of job j, $j = 1, \ldots, n$.

$F \mid\mid C_{max}$ is strongly NP-hard [5], even for instances with only three machines ($F3||C_{max}$). However, Johnson [7] developed a solution method that finds the optimal solution for the $F2||C_{max}$ problem in polynomial time. When it comes to solution approaches, a variety of methods have been developed for different flow shop scheduling problems. Johnson's rule [7] has served as a basis for developing heuristic methods for general flow shop scheduling problems. Due to the complexity of the flow shop scheduling problem, the main focus has been on developing heuristic algorithms.

The Shifting Bottleneck (SB) algorithm, developed by Adams et al. [1] is among the most popular and powerful heuristic solution method for the Job Shop Scheduling Problem ($J||C_{max}$). SB is an iterative algorithm that decomposes the job shop scheduling problem into single machine scheduling subproblems (i.e., the single machine scheduling problem with release dates and due dates, where the objective is to minimize the maximum lateness, $1|r_j|L_{max}$). These single machine scheduling problems are then solved and the machine with the largest maximum lateness, called the *bottleneck machine*, is given priority in the schedule. The performance (both the quality of solution and the computational complexity) of the SB algorithm is highly dependent on the algorithm which is used for solving the subproblems. $F||C_{max}$ is a special case of the job shop scheduling problem, where the order in which the jobs must pass through machines is identical for all jobs. Therefore, the SB heuristic could be a promising method for solving flow shop scheduling problems. We focus on the SB method because it is suitable for parallelization and it will allow us to exploit the huge computing power of the current and future parallel computing systems such as multi-core and many-core systems. In this research, we design two efficient parallel algorithms based on the SB heuristic for $F||C_{max}$. The first one is a *coarse-grained parallel algorithm* that is suitable for execution on multi-core systems with a small number of cores, while the second one is a *fine-grained parallel algorithm* suitable for multi-core systems with a large number of cores. To design the parallel algorithms we leverage our previous work [4] on designing parallel randomized approximation algorithms for $1|r_j|L_{max}$. We perform an extensive experimental analysis on instances of various sizes to evaluate the performance of the proposed parallel algorithms for $F||C_{max}$. To the best of our knowledge the proposed algorithms are the first parallel algorithms for solving the $F||C_{max}$ problem presented in the literature.

3.1 Our proposed algorithms

Our proposed parallel SB algorithms (PSB-CG and PSB-FG) for the $F||C_{max}$ problem are presented in Algorithm 1 as a single algorithm called PSB-X, where X can be CG or FG. The input to PSB-X consists of the number of machines m, the number of jobs n, the processing times of the operations t_{ij}, and the binary variable $REOPT$, which specifies whether a re-optimization policy is used or not. In the first step (line (1)) the algorithm initializes the set of scheduled machines, $\widehat{\Sigma}$, to the empty set. In lines (2)-(10), it iteratively schedules the bottleneck machine until all the machines are scheduled (i.e., $\widehat{\Sigma} = \Sigma$, where Σ is the initial set of machines). In each iteration, it finds the longest path in graph G, and updates the value of release dates, r_{ij}, and due dates, d_{ij}, of all operations. In lines (5)-(10), it identifies the bottleneck machine from the set of unscheduled machines, $\Sigma - \widehat{\Sigma}$, and schedules it. These steps are executed in parallel in both PSB-CG and PSB-CG algorithms. To identify the bottleneck machine, the algorithm solves the $1|r_j|L_{max}$ problem for each unscheduled machine.

The PSB-CG algorithm, calls the serial version of the randomized approximation algorithm (RAA-S) to solve the single machine scheduling problems and identifies the bottleneck machine. The PSB-FG algorithm, employs RAA-P to solve each single machine scheduling problem using multiple cores. Once the single machine scheduling problems are solved for all unscheduled machines, in line (7) it determines the machine with the maximum lateness, L_{max}, as the bottleneck machine. Determining the machine with the maximum lateness is done in parallel by using a parallel reduction operation with minimum as the reduction operator [8]. In line (9), it updates the graph G by adding the disjunctive arcs corresponding to the solution of the bottleneck machine. Then, in line (10), it adds the bottleneck machine to the set of

Algorithm 1 Parallel Shifting Bottleneck Algorithms (PSB-X)

```
Input: $n$, $m$, $t_{ij}$, $j = 1, \dots, n$, $j = 1, \dots, m$, $REOPT$
 1: Set $\widehat{\Sigma} = \emptyset$.
 2: while $\Sigma \neq \widehat{\Sigma}$ do
 3:     $C_{max}(\widehat{\Sigma}) \leftarrow$ the longest path in disjunctive graph $G$.
 4:     Compute $r_{ij}$ and $d_{ij}$ of all operations.
 5:     for all machines in $\Sigma \setminus \widehat{\Sigma}$ do in parallel
 6:         Call RAA-S for PSB-CG or
             Call RAA-P for PSB-FG
 7:         $\beta \leftarrow \arg\max_{i \in (\Sigma \setminus \widehat{\Sigma})} L_{max}(i)$
 8:     end for
 9:     Add disjunctive arcs corresponding to the sequence of machine $\beta$ in graph $G$.
10:     $\Sigma \leftarrow \Sigma \setminus \{\beta\}$ and $\widehat{\Sigma} \leftarrow \widehat{\Sigma} \cup \{\beta\}$.
11:     if $REOPT = True$ then
12:         for all machines in $\widehat{\Sigma} \setminus \{\beta\}$ do
13:             Remove the disjunctive arc corresponding to the schedule of the machine.
14:             Call RAA-S for PSB-CG or
                 Call RAA-P for PSB-FG
15:             if there is an improvement in $C_{max}$ then
16:                 Add disjunctive arcs corresponding to the new schedule in graph $G$.
17:             else
18:                 Restore disjunctive arcs removed in line (13).
19:             end if
20:         end for
21:     end if
22: end while
```

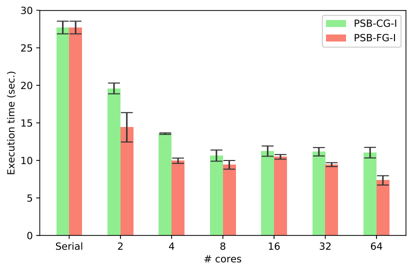
scheduled machines, $\widehat{\Sigma}$, and removes it from Σ. Lines (11)-(22) are associated with the re-optimization of the scheduled machines. In fact, these steps are optional in the SB heuristic and are executed only if $REOPT = True$.

There are different policies one might consider for re-optimization. In this research, we employ a full re-optimization policy, that is, all the machines in set $\widehat{\Sigma}$ excluding the last scheduled machine are considered for re-optimization. Based on this policy, in each iteration the algorithm removes the disjunctive arcs corresponding to the solution of an already scheduled machine (line (13)), then it solves $1|r_j|L_{max}$ for that machine. In the case of the PSB-CG algorithm, the single machine scheduling problem is solved using the serial approximation algorithm RAA-S, while in the case of PSB-FG it is solved using the parallel approximation algorithm RAA-P. Here, the machines are re-optimized based on the order in which they have been chosen as bottlenecks. In lines (15)-(18), the algorithm adds to G the disjunctive arcs corresponding to the new schedule of the single machine if this schedule improves the makespan, otherwise it restores the previous disjunctive arcs that were removed in line (13). The algorithm that we use to solve the subproblems is a randomized approximation algorithm proposed in Section 2.

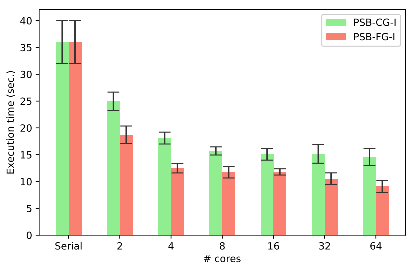
3.1.1 Experimental analysis

Here, we present a summary of our experimental analysis and consider large size instances for which the serial version of the proposed parallel algorithms might not be able to obtain solutions in a reasonable amount of time. We solve these instances using the two proposed parallel algorithms (i.e., PSB-CG and PSB-FG). The large-size instances range from 6×10^3 to 25×10^3. All the experiments for large-size instances are performed with $\epsilon = 0.8$. We denote by PSB-CG-I and PSB-FG-I, the parallel algorithms in which no re-optimization policy is used, and by PSB-CG-II and PSB-FG-II, the parallel algorithms in which the full re-optimization policy is used.

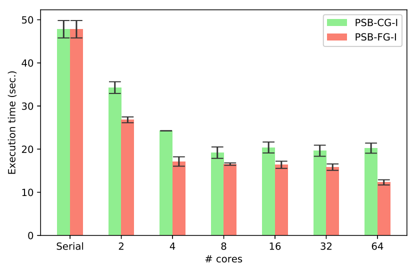
The execution times of PSB-CG-I and PSB-FG-I for the instances ranging from $(5, 3\times10^3)$ to $(5, 5\times10^3)$



(a) $n = 3000$

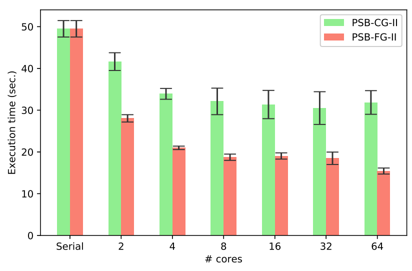


(b) $n = 4000$

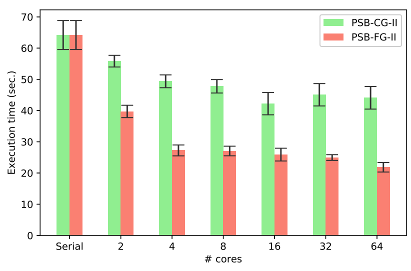


(c) $n = 5000$

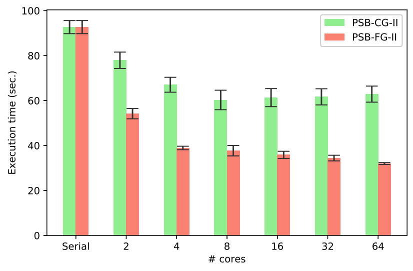
Figure 3: PSB-CG-I and PSB-FG-I (without re-optimization): Execution time vs. number of cores for large-size instances ($m = 5, \epsilon = 0.8$)



(a) $n = 3000$



(b) $n = 4000$



(c) $n = 5000$

Figure 4: PSB-CG-II and PSB-FG-II (with re-optimization): Execution time vs. number of cores for large-size instances ($m = 5, \epsilon = 0.8$)

Table 3: Best average execution times obtained by the parallel algorithms (sec.)

m	n	PSB-CG-I	PSB-CG-II	PSB-FG-I	PSB-FG-II
3	2000	4.161	7.576	2.034	3.366
	3000	6.777	12.975	2.882	5.578
	4000	9.411	16.960	4.123	7.515
	5000	11.750	24.272	5.201	10.495
5	2000	6.371	18.756	4.650	9.311
	3000	10.630	30.504	7.333	15.426
	4000	14.554	42.226	9.103	21.831
	5000	19.189	60.274	12.299	31.965

are provided in Figure 3. In these algorithms, there is no re-optimization on the machines that have already been scheduled. We observe that performing the full re-optimization policy (Figure 4) affects the efficiency of the parallel algorithm. We also observe that the execution time of PSB-CG-II is more sensitive to the full re-optimization policy than PSB-FG-II. For example, for the instance $(5, 5 \times 10^3)$, the best execution times of PSB-CG-I and PSB-CG-II are 19.189 and 60.274 seconds, respectively. While for the same size of instances, PSB-FG-I and PSB-FG-II take 12.299 and 31.965 seconds, respectively. This difference stems from the fact that the re-optimization part of the algorithms is completely performed serially in the PSB-CG-II algorithm, while in PSB-FG-II the re-optimization of each machine is preformed using the parallel randomized approximation algorithm RAA-P.

Table 3 presents the best average execution times obtained by the PSB-CG-I, PSB-CG-II, PSB-FG-I, and PSB-FG-II algorithms for the large-size instances. We observe that the proposed algorithms can solve flow shop problem instances up to size $(5, 5 \times 10^3)$ in less than a minute. This table also shows that the execution time of the proposed algorithms is not considerably sensitive to the number of jobs, that is, we can solve instances with large number of jobs in a reasonable amount of time. These results confirm the efficiency of the proposed algorithms for large-size flow shop problems. For small and medium-size instances, we observed that the value of ϵ significantly affects the execution time of the proposed algorithms. The same behavior is expected for large-size instances. Therefore, in practice, we need to choose a proper value for ϵ according to our expectations of the quality of solution.

The experimental analysis showed that the proposed parallel algorithms obtain high quality solutions for fairly large-size instances of the flow shop scheduling problem in a reasonable amount of time. With the fine-grained parallel algorithm one can solve instances with 5 machines and 5×10^3 jobs in less than a minute. We also observed that the full re-optimization policy can improve the quality of solutions significantly for instances of all sizes.

References

[1] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[2] Hossein Badri, Tayebeh Bahreini, and Daniel Grosu. A parallel randomized approximation algorithm for non-preemptive single machine scheduling with release dates and delivery times. *Submitted to Computers & Operations Research.*

[3] Hossein Badri, Tayebeh Bahreini, and Daniel Grosu. Parallel shifting bottleneck algorithms for flow shop scheduling. *Submitted to IEEE Transactions on Systems, Man and Cybernetics: Systems.*

[4] Hossein Badri, Tayebeh Bahreini, and Daniel Grosu. A parallel randomized approximation algorithm for the non-preemptive single machine scheduling problem. *WSU CS Technical Report*, 2018.

[5] Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

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

[7] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics (NRL)*, 1(1):61–68, 1954.

[8] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2003.

[9] E. Lawler, J.K Lenstra, A.H.G Rinnooy Kan, and D.B Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C Graves, A.H.G Rinnooy Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science: Logistics of Production and Inventory.* North-Holland, Amsterdam, 1993.

[10] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[11] Eugeniusz Nowicki and Czesław Smutnicki. An approximation algorithm for a single-machine scheduling problem with release times and delivery times. *Discrete Applied Mathematics*, 48(1):69–79, 1994.

[12] CN Potts. Technical note—analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28(6):1436–1441, 1980.

[13] L Schrage. Obtaining optimal solutions to resource constrained network scheduling problems. *Unpublished manuscript*, 189, 1971.

[14] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.