Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Industrial Engineering 47 (2004) 123–140

computers & industrial engineering

www.elsevier.com/locate/dsw

Minimizing total flow time in single machine environment with release time: an experimental analysis☆

Y. Guo[a], A. Lim[b,*], B. Rodrigues[c], S. Yu[a]

[a]*School of Computing, National University of Singapore, 3 Science Drive 2, Singapore, Singapore 117543*
[b]*Department of Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, China*
[c]*School of Business, Singapore Management University, 269 Bukit Timah Road, Singapore, Singapore 259756*

Received 3 April 2004; accepted 14 June 2004
Available online 28 July 2004

Abstract

We study the problem of minimizing total flow time on a single machine with job release times. This problem is **NP**-complete for which is no constant ratio approximation algorithm. Our objective is to study experimentally how well, on average, the problem can be solved. The algorithm we use produces non-preemptive schedules converted from preemptive ones. We evaluate average solution quality for the problem to identify the characteristics of difficult instances. Results obtained are compared with those recently obtained by other researchers. Based on extensive experiments, we also develop an empirical model to predict solution quality using interpolation.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: Machine scheduling; Flow time; Experimental analysis; Performance prediction

1. Introduction

Extensive studies have been carried out on scheduling jobs on a single machine with various objective functions. These problems are important because single machine environments are common and a special case of other environments. In this work, we consider a basic, yet intractable problem from the worst-case point of view—the problem of minimizing total flow time in a single machine environment with job release times. In this problem, there are n independent jobs to be scheduled non-preemptively

☆ Processed by Area Editor Maged M. Dessouky.
* Corresponding author.
E-mail address: iealim@ust.hk (A. Lim).

doi:10.1016/j.cie.2004.06.004

on a single machine. Each job has a release time r_j, indicating the earliest possible time that the job can be processed, a processing time p_j, and a completion time C_j. The objective is to minimize the sum of flow times of all jobs, where the flow time, F_j, of a job j, is defined to be the time elapsed from its release to its completion, i.e. $F_j=(C_j-r_j)$. In machine-scheduling notation (Pinedo, 1995), the problem is denoted by $1|r_j|\sum F_j$. In systems involving queuing, in networks, for example, the flow time of a job consists of both the waiting time in the queue and the job processing time on the machine so that minimizing flow time improves service quality (Leonardi & Raz, 1997). The $1|r_j|\sum F_j$. problem has many applications, for example, in network systems and parallel computing (Kellerer, Tautenhahn, & Woeginger, 1996).

If all jobs are released at the same time, the problem reduces to the $1|r_j|\sum F_j$. problem, which can be solved optimally by the well-known Shortest Processing Time (SPT) rule (Smith, 1956). The preemptive version, $1|\text{pmtn}, r_j|\sum F_j$, of the problem can be solved optimally in polynomial time by the Shortest Remaining Processing Time (SRPT) rule, as shown by Baker (1974). The solution of the preemptive problem provides a lower bound for the objective value of the non-preemptive problem and Ahmadi and Bagchi (1990) have shown that this lower bound dominates all other known lower bounds for the $1|r_j|\sum F_j$ problem. With arbitrary release times, the problem becomes **NP**-complete (Lenstra, Rinnooy Kan, & Brucker, 1977). Some authors (Chandra, 1979; Chu, 1992b; Deogun, 1983; Dessouky & Deogun, 1981) have developed branch-and-bound algorithms and Chu (1992a), and Mao and Rifkin (unpublished) have proposed approximation algorithms for $1|r_j|\sum F_j$ problem.

The Earliest Start Time (EST) rule assigns shortest available jobs to a machine when they are released and the Earliest Completion Time (ECT) rule assigns the job (which may not be available yet) with earliest possible completion time to a machine. Both rules have a worst-case bound of $O(n)$. Kellerer et al. (1996) developed an approximation algorithm with a sublinear worst-case performance guarantee of $O(\sqrt{n})$. They also proved that unless $\mathbf{P}=\mathbf{NP}$, there exists no polynomial time approximation algorithm for $1|r_j|\sum F_j$ with a worst-case performance bound of $O(n^{1/2-\varepsilon})$ for any $\varepsilon>0$. In other words, no constant ratio approximation algorithm can be expected for the problem.

Another closely related **NP**-complete problem is $1|r_j|\sum C_j$. We note that an optimal solution for $1|r_j|\sum C_j$ is also an optimal solution for $1|r_j|\sum F_j$, because for any instance of the problem, $\sum r_j$ is a constant, and minimizing $\sum F_j=\sum(C_j-r_j)=\sum C_j-\sum r_j$ is equivalent to minimizing $\sum C_j$. Recently, researchers have achieved better results for $1|r_j|\sum C_j$ when compared with the $1|r_j|\sum F_j$ problem. For example, Phillips, Stein, and Wein (1995) developed a $(16+\varepsilon)$-approximation algorithm for the weighted version $1|r_j|\sum w_jC_j$. Hall, Shmoys, and Wein (1996) improved the ratio to 4. Because of the closely related objectives, new algorithms for $1|r_j|\sum F_j$ can be devised with algorithms for $1|r_j|\sum C_j$. Phillips, Stein, and wein (1998) developed a 2-approximation algorithm for the $1|r_j|\sum C_j$ problem which we call the PSW algorithm. This algorithm first produces preemptive schedules which are optimal and then converts these schedules to non-preemptive ones. This method is commonly used to produce non-preemptive schedules. In this study, we modify the PSW algorithm to solve the $1|r_j|\sum F_j$ problem. We ascertain whether, on average, this new algorithm yields good solutions for the $1|r_j|\sum F_j$ problem and determine situations in which performance is unsatisfactory.

The modified algorithm produces non-preemptive schedules from preemptive ones. Given a set of jobs with release times and processing times, we first form a preemptive schedule using the SRPT rule. Under this rule, the machine always picks jobs with the shortest remaining processing times among those already released at the current time and processes these first. Each job will have a (preemptive) completion time C_j^P. Next, we form an ordered list L of jobs based on their preemptive completion time C_j^P using a simple

```
The Modified-PSW Algorithm

Input: A set of jobs with release time and processing time
Output: A non-preemptive schedule S

    1. Form a preemptive schedule by SRPT, and record the completion time
       of each job as $C_j^P$.
    2. Form an increasing ordered list L by sorting $C_j^P$.
    3. While L is not empty {
            3.1 Assign the first job on L to the machine
            3.2 Delete the assigned job from L
       }
```

Fig. 1. An outline of algorithm modified-PSW.

sort. A non-preemptive schedule is then obtained if we continue to assign the first job in L to the machine when it is freed and delete it from L. We call this algorithm the modified-PSW algorithm, which is outlined in Fig. 1. If we let C_j^{o}, C_j^{m}, F_j^{o}, F_j^{m} denote the completion times and flow times of a job in the optimal schedule and the schedule given by the modified-PSW algorithm, respectively, it is easy to see that $C_j^{\mathrm{m}}/C_j^{o} \le 2$ (Phillips et al., 1998) but that $F_j^{\mathrm{m}}/F_j^{\mathrm{o}} = (C_j^{\mathrm{m}} - r_j)/(C_j^{\mathrm{o}} - r_j) \le 2$ does not necessarily hold.

2. Experiment design

In order to conduct systematic experiments, we describe here a performance metric, parameters, factors and levels and experiment procedure. We first choose a performance metric, parameters, factors and their levels, and then describe the experiment procedure (Jain, 1991).

Performance Metric. To test whether the modified-PSW performs well, we use preemptive schedules for $1|r_j, \mathrm{pmtn}|\sum F_j$ as a lower bounds. Let F_j^L denote the flow time of a job in the lower bound. The performance metric *Ratio* is defined to be the deviation of modified-PSW from the lower bound, i.e. $\mathrm{Ratio} = \left(\sum_j F_j^{\mathrm{m}} - \sum_j F_j^L\right)/\sum_j F_j^L$

Parameters. In the $1|r_j|\sum F_j$ problem, there are three parameters that can influence the performance metric. The first is the number of jobs n, which can reach hundreds of thousands. Kellerer et al. (1996) proposed a lower bound related to n for the problem. The second is job release times. On one extreme, when all jobs are released in a very small interval, the problem is reduced to $1||\sum F_j$, and can be solved optimally. On the other extreme, when the jobs are released very slowly, a simple Earliest Released Date First rule will also solve the problem optimally. Between these extremes, performance is not expected to be as good. Without loss of generality, we assume that the smallest release time is 0. In addition, we also assume that the job release time follows a uniform distribution. Thus, the second parameter is reduced to the maximum release time of all jobs, denoted by R. The third parameter is the mean processing time of jobs, μ. We study the cases where the processing time is exponentially and uniformly distributed.

Factors and levels. As the purpose here is to study properties of the modified-PSW algorithm under different scheduling environments, and there are only three parameters, we will choose all three parameters above to be factors in the experiments. Table 1 summarizes the factors and levels used in our following study.

Experiment procedure. This study will focus on the average performance of modified-PSW algorithm in solving $1|r_j|\sum F_j$. We first study how *Ratio* varies with n, R, and μ, respectively. At every level of

Table 1
Factors and their corresponding distributions and levels

Factor	Distribution	Level
n		10, 50, 100, 500, 1000, 5000, 10,000, 50,000, 100,000
R	Uniform	100, 500, 1000, 5000, 100,00, 50,000, 100,000, 500,000, 1,000,000
μ	Exponential, uniform	1, 5, 10, 50, 100, 500, 1000, 5000, 10,000

a given factor, 100 test cases are generated randomly. Then, based on the results, we increase the number of levels appropriately, and conduct further experiments to test the performance of modified-PSW when two factors are allowed to vary simultaneously. Fifty test cases are generated for each combination of factor levels and an empirical model is developed to predict the average performance of modified-PSW for combinations of the three factors. The accuracy of the model is tested against n, R and μ.

The necessity for experiment. As theoretical bounds have already been derived by Kellerer et al. (1996), the main purpose here is to study how, when compared with these bounds, the modified-PSW algorithm performs, on average, for $1|r_j|\sum F_j$ under different parameter combinations. Theoretical analysis focuses only on the worst-case. Here, we are motivated by the applicability of the modified-PSW to problems in practice. Further, since different values of parameters affect performance to different extents, we implement experiments to determine how the modified-PSW algorithm responds to parameter combinations.

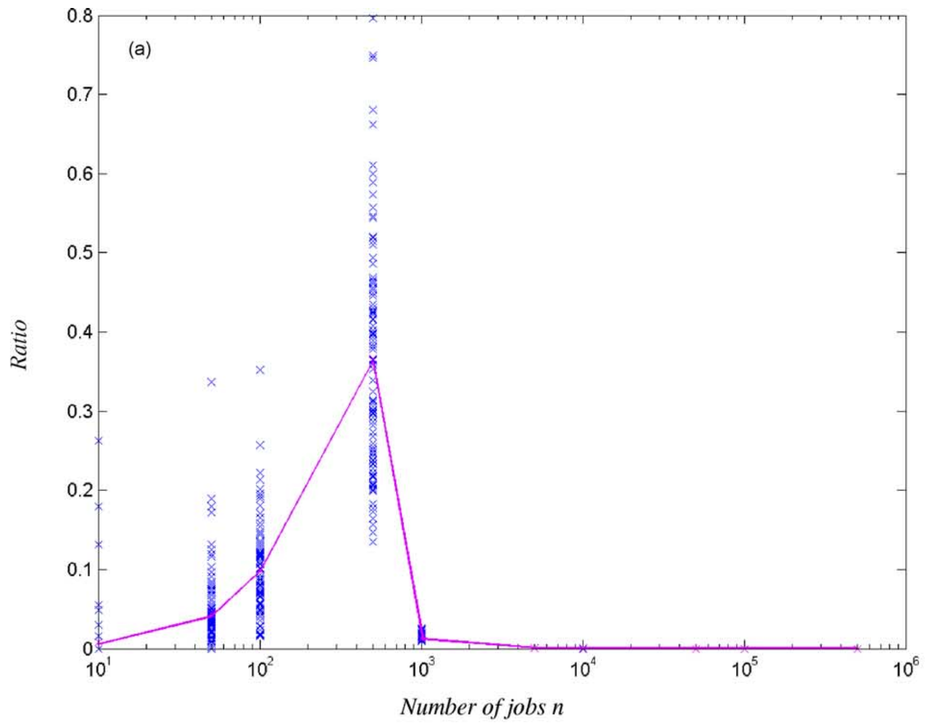
Reproducibility. Since reproducibility is crucial to any experimental study, we describe here the Random Number Generator that is used in our test case generation. A Java package, COLT, consisting of six libraries for scientific and technical computing was used to generate numbers from exponential and normal distributions (http://hoschek.home.cern.ch/hoschek/colt/V1.0.3/doc/index.html). We used Ranecu to produce exponentially distributed random numbers for job processing time. Ranecu is an advanced multiplicative linear congruential random number generator with a period of approximately 10^{18}. We used MersenneTwister to produce uniformly distributed random numbers for job release time and for job processing time. All test cases were run on a Pentium III (600 HZ) processor and 128 MB memory and algorithms coded in Java and compiled and run with Java sdk1.4.0 and COLT1.0.3.

3. One-factor experiments

In this section, we describe one-factor experiments, which reveal how n, R and μ affect performance individually. One-factor experiments are insightful to the nature of the problem (e.g. whether the performance is convex, concave, monotonically increasing or decreasing with each parameter). Further, the results of one-factor experiments suggest directions for two-factor experiments.

3.1. Testing the effect of number of jobs on the results

To determine the effect of n on *Ratio*, we fixed R at 5000 and μ at 10 for exponentially distributed processing time First, we varied n on a log scale (as shown in Table 1) to obtain a general picture of the average performance within a large range of n. Fig. 2(a) is a plot of *Ratio* against n. The ‘X’ marks are the results of the modified-PSW algorithm on test instances with the given parameter settings,



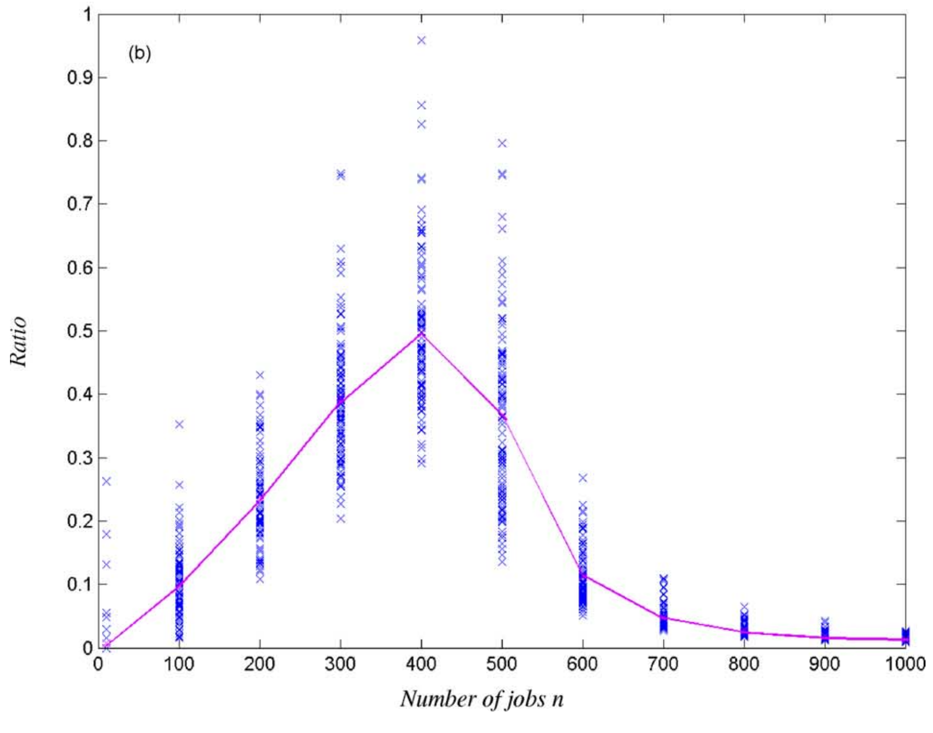


Fig. 2. (a) Plot of *Ratio* against *n*; (b) Plot of *Ratio* against *n* in the range [0, 1000].

and the lines connect averages. We observe that the average performance is satisfactory when n is less than 100 where *Ratio* is clustered between 10 and 20%. The standard deviations at these n values were not small, however, where, for example, the largest *Ratio* value of about 0.35 occurred when the average *Ratio* value was below 0.1. In the range of n between 100 and 1000, performance is poor as values exceed 0.2 and *Ratio* reaches 0.8. Also, the standard deviation becomes large here. When n is larger than 1000, performance improves where the absolute values of *Ratio* are very close to 0 and standard deviations are small. When we look in detail, we found that the worst average occurs when $n=400$ and performance is satisfactory when n exceeds 700. Fig. 2(b) is a plot of *Ratio* against n on a linear scale in the range [0, 1000]. From these plots, we see that modified-PSW algorithm performs well, on average, when n is less than 100 or n is greater than 700 (where R and μ are set to 5000 and 10, respectively), where smaller *Ratio* values are found for test sets with smaller standard deviations.

3.2. Testing the effect of maximum job release time R on results

To determine the effects of R on *Ratio*, we fixed n at 500 and μ at 10 for exponentially distributed processing time First, we varied R on log scale. Fig. 3(a) plots *Ratio* against R as R increases from 10^2 to 10^6. Performance is satisfactory when R is less than 10^3 or larger than 10^5. Between 10^3 and 10^5, performance is poor with *Ratio* exceeding 0.2. The standard deviations of *Ratio* were proportional to its absolute values. Fig. 3(b) and (c) provides magnified plots for R in the ranges of (10^3, 10^4) and (10^4, 10^5), respectively. The worst average case occurred when R was 6000 with a *Ratio* value near 0.5. From these three plots, we see that the modified-PSW performs well, on average, when R is less than 3000 or larger than 50,000. We found that the more difficult test sets were those with R near to 10^4. Similar to the case for n, smaller *Ratio* values usually occurred with smaller standard deviations.

3.3. Testing the effect of μ on results

Finally, to test the effects of μ (for exponentially distributed processing time) on *Ratio*, we fixed n at 500 and R at 5000. First we varied μ on log scale. Fig. 4(a) plots *Ratio* against μ as μ varies from 1 to 10^4 on a log scale. The performance is satisfactory when μ is greater than 50, and is relatively poor when μ is smaller than 50. Fig. 4(b) provides a zoom-in linear-scale plot at the μ range of (1, 20). The worst average case occurs at $\mu=7$ and *Ratio* is around 0.4 but the largest standard deviation is at $\mu=10$. From Fig. 4, we infer that the modified-PSW algorithm performs well on average when μ is greater than 20. Similar to the cases for n and R, smaller *Ratio* values occurred with smaller standard deviations and at these points the average case performance was good. Further experiments were carried out for jobs with uniformly distributed processing time. The resulting plots are very close to those given in Figs. 2–4, except that the worst average *Ratio* was reduced from approximately 0.5 to about 0.20.

4. Two-factor experiments

Having determined from one-factor experiments that performance is approximately a convex function of each parameter n, R, and μ individually, we investigated the combined effects of these factors to provide a better understanding of the impact of problem characteristics on the algorithm used.

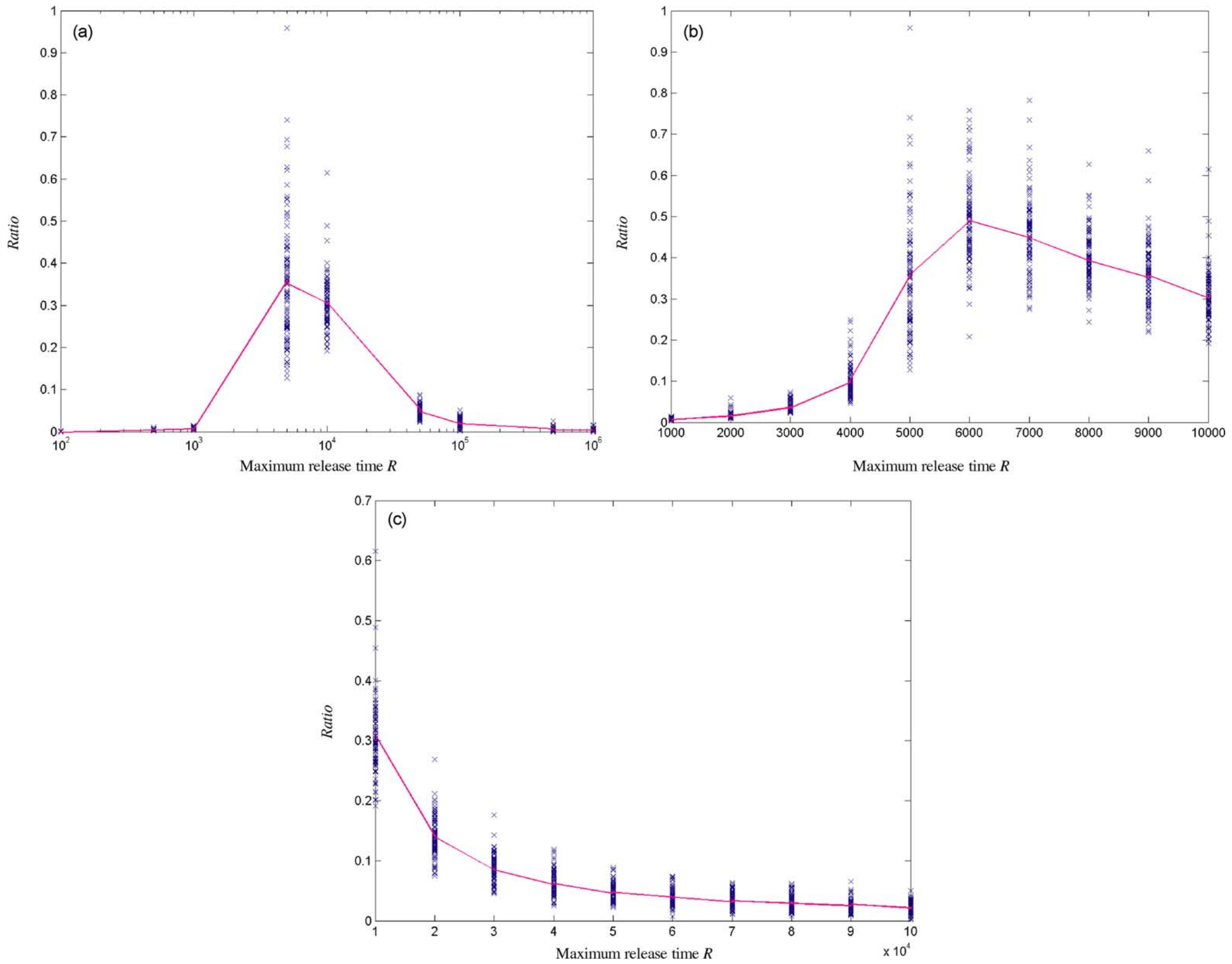
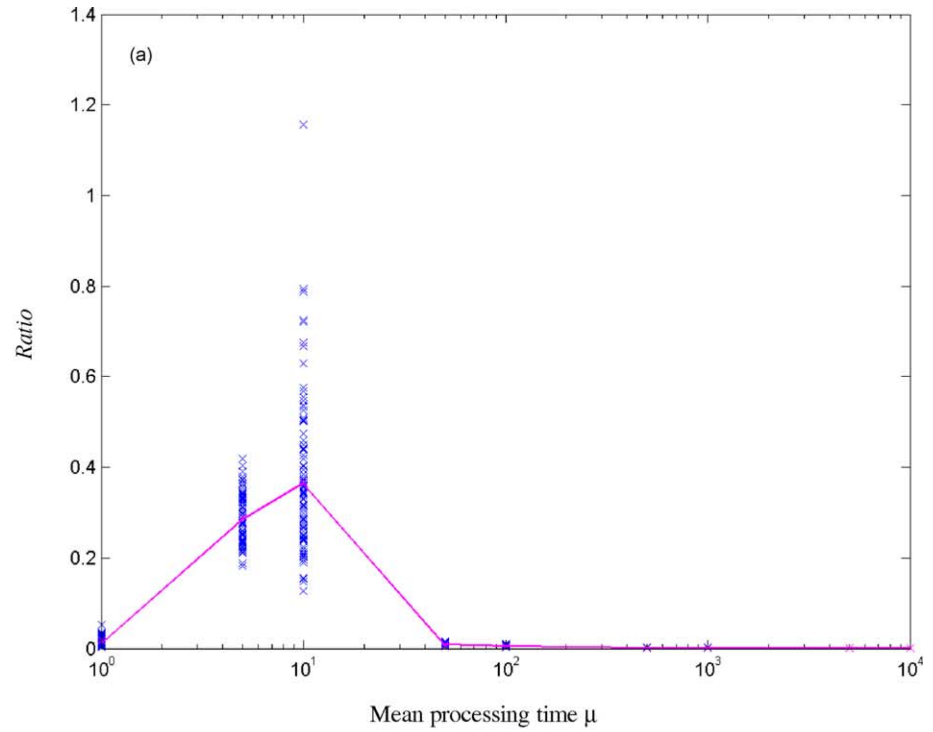


Fig. 3. (a) Plot of *Ratio* against *R*; (b) Plot of *Ratio* against *R* in the range [1000, 10,000]; and (c) Plot of *Ratio* against *R* in the range [10,000, 100,000].



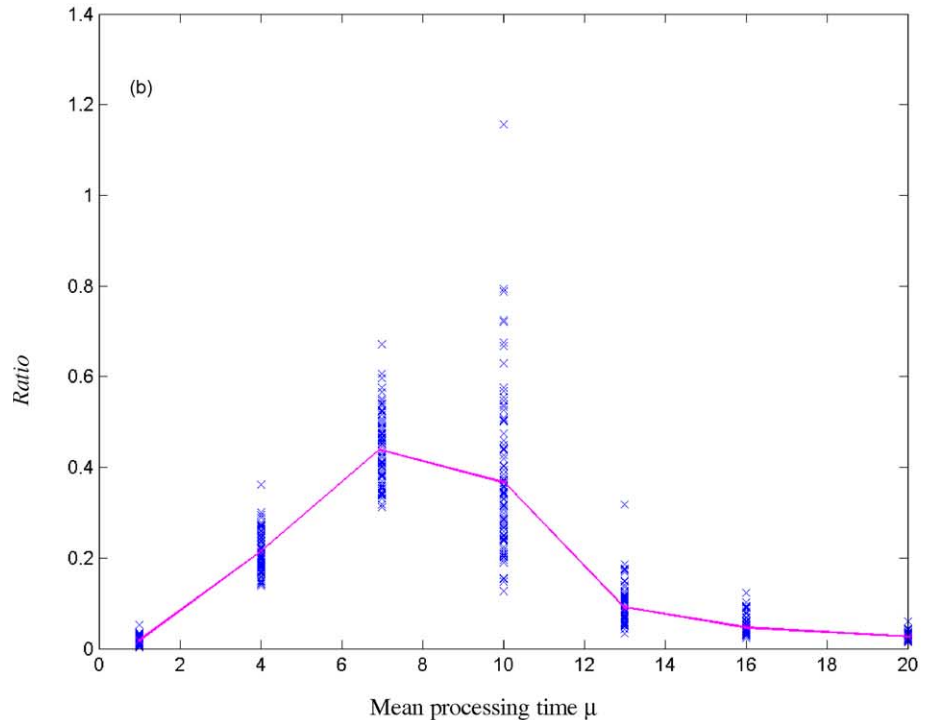


Fig. 4. (a) Plot of *Ratio* against μ; (b) Plot of *Ratio* against μ in the range [0, 20].

4.1. Reduction of the μ factor

When we examined the relationship between R and μ, we noticed these factors were related We first give two lemmas. Let X and Y denote the random variables for a job's release time and processing time, respectively.

Lemma 1. *If X is uniformly distributed $U(0, R)$, then $\alpha(X/\mu)$ is uniformly distributed $U(0, \alpha(R/\mu))$, where $\alpha > 0$ is a constant.*

Proof. X is uniformly distributed $U(0,R) \Leftrightarrow Pr\{X < r) = r/R$.

Hence,

$$\Pr\left\{\frac{\alpha X}{\mu} < r\right\} = \Pr\left\{X < \frac{\mu r}{\alpha}\right\} = \mu r/\alpha R = r/(\alpha R/\mu) \tag{1}$$

Since α is a constant, Eq. (1) indicates that $\alpha(X/\mu)$ is uniformly distributed $U(0, \alpha(R/\mu))$. □

Lemma 2. *If Y is exponentially distributed $Exp(\mu)$, then $\alpha(Y/\mu)$ is exponentially distributed $Exp(\alpha)$, where $\alpha > 0$ is a constant.*

Proof. Y follows an exponential distribution $\mathrm{Exp}(\mu) \Leftrightarrow P\{Y < p) = 1 - \mathrm{e}^{-p/u}$

Hence

$$P\left\{\frac{\alpha Y}{\mu} < p\right\} = P\left\{Y < \frac{\mu p}{\alpha}\right\} = 1 - e^{-(\mu p/\alpha)/\mu} = 1 - e^{-p/\alpha} \tag{2}$$

Since α is a constant, Eq. (2) implies that $\alpha(Y/\mu)$ is exponentially distributed $Exp(\alpha)$. □

We can now state and prove the following theorem.

Theorem 1. Any instance of $1|r_j|\sum F_j$ with factors $\{n, R, \mu\}$ can be transformed into an equivalent instance with factors $\{n, \alpha(R/\mu), \alpha\}$, where α is a positive constant.

Proof. First we note that by transforming R to $\alpha(R/\mu)$ and μ to α, the processing time and the release time of every job are scaled by α/μ. Now, R and μ are the only time-dependant parameters. Therefore, the change of μ to α and R to $\alpha(R/\mu)$ is a change of the time unit from 1 to (α/μ). If we let S_1 denote the schedule for an instance $\{n, R, \mu\}$, and S_2 denote the schedule for the instance $\{n, \alpha(R/\mu), \alpha\}$, then since the modified-PSW algorithm produces schedules based on the relative values of job release times and processing times, the relative order of jobs in S_1 and S_2 must be the same, and therefore, *Ratio* values resulting from S_1 and S_2 should have the same distribution. In other words, an instance with factors $\{n, R, \mu\}$ and its transformed instance with factors $\{n, \alpha(R/\mu), \alpha\}$ are equivalent in terms of average solution quality. Finally, the transformations R to $\alpha(R/\mu)$ and μ to α results in uniform and exponential distributions, respectively, from the Lemmas 1 and 2. □

4.2. Testing the combined effects of n and R on Ratio

We now study the combined effects of n and R on the average performance. The situation is more complicated than the one-variable tests studied in previous sections for which we use 3D graphs to illustrate test results. We used 44 different n values and 54 R values, with a total of $44*54=2376$ (n, R) pairs, and for each (n, R) pair, there were 50 randomly generated test cases. However, the n and R values were not uniformly distributed since, from the 2D plots, we know more test sets are selected for those n and R values where *Ratio* is greater than 0.1 and where the slope of the 2D plot is large. The 3D plot represents the approximate shape of *Ratio* distributions. Different views of the same 3D plot are shown where the mean processing time of jobs μ is fixed at 10.

Fig. 5(a)–(c) show plots on linear scale from three different views. The largest *Ratio* value is about 0.45, whereas the lowest is in the range of 0–0.05. We observe that when we draw a bottom-left to top-right diagonal on the base plane (Fig. 5(c)), we find that the points that are sufficiently far from the diagonal have *Ratio* values smaller than 0.1, and points nearer to the diagonal have larger *Ratio* values, mostly above 0.2.

From Fig. 6(a)–(c), which are the same plots on log scale from different views as Fig. 5(a)–(c), we make several other observations. First, there is a ridge-like shape in the graph (Fig. 6(b)). There is a large area of flat and low areas, where the average performance is good, a long and high ridge, where the average performance is poor and a steep slope connecting the low areas to the ridge. The projection of the ridge onto the base plane is a straight line, which coincides with the diagonal (Fig. 6(c)). This suggests that when $\log n$ and $\log R$ satisfy the linear relationship $\log n=k\log R+b$, the average performance is likely to be poor. To explain this, we hypothesize that the average performance is closely related to the number of preemptions during the formation of a preemptive schedule. Intuitively, more frequent preemptions imply a bigger gap between the preemptive and the non-preemptive schedules. On one extreme, when n is so small that $\log n \ll k\log R+b$, the machine is lightly loaded and there are few simultaneous outstanding jobs, so jobs are likely to be processed in first-come-first-serve order and thus the number of preemptions is small. On the other extreme, when n so large that $\log n \gg k\log R+b$, the machine is heavily loaded and there are a large number of simultaneous outstanding jobs, which is analogous to the case when all jobs are released at the same time, so that SRPT performs as well as SPT and the number of preemptions is small. Between these extremes, machine utilization is moderate but a bigger number of preemptions occur in the optimal preemptive schedule.

In order to investigate the variation of the test results in the modified-PSW algorithm, we plotted a 3D graph for the standard deviation of the test results in Fig. 7.

This plot also has a ridge which is lower compared with those found before. The maximum standard deviation is approximately 0.2, and for flat areas, standard deviations are less than 0.05.

Let k be a positive constant, we can infer the following from the experimental analysis:

(1) For $n=o(R^k)$ or $n=\omega(R^k)$, the average performance is expected to be within 0.1 from the optimal, and the standard deviation is less than 0.05.
(2) For $n=\Theta(R^k)$, the average performance is expected to be from 0.1 to 0.5 from the optimal, and the standard deviation varies from 0.05 to 0.2.

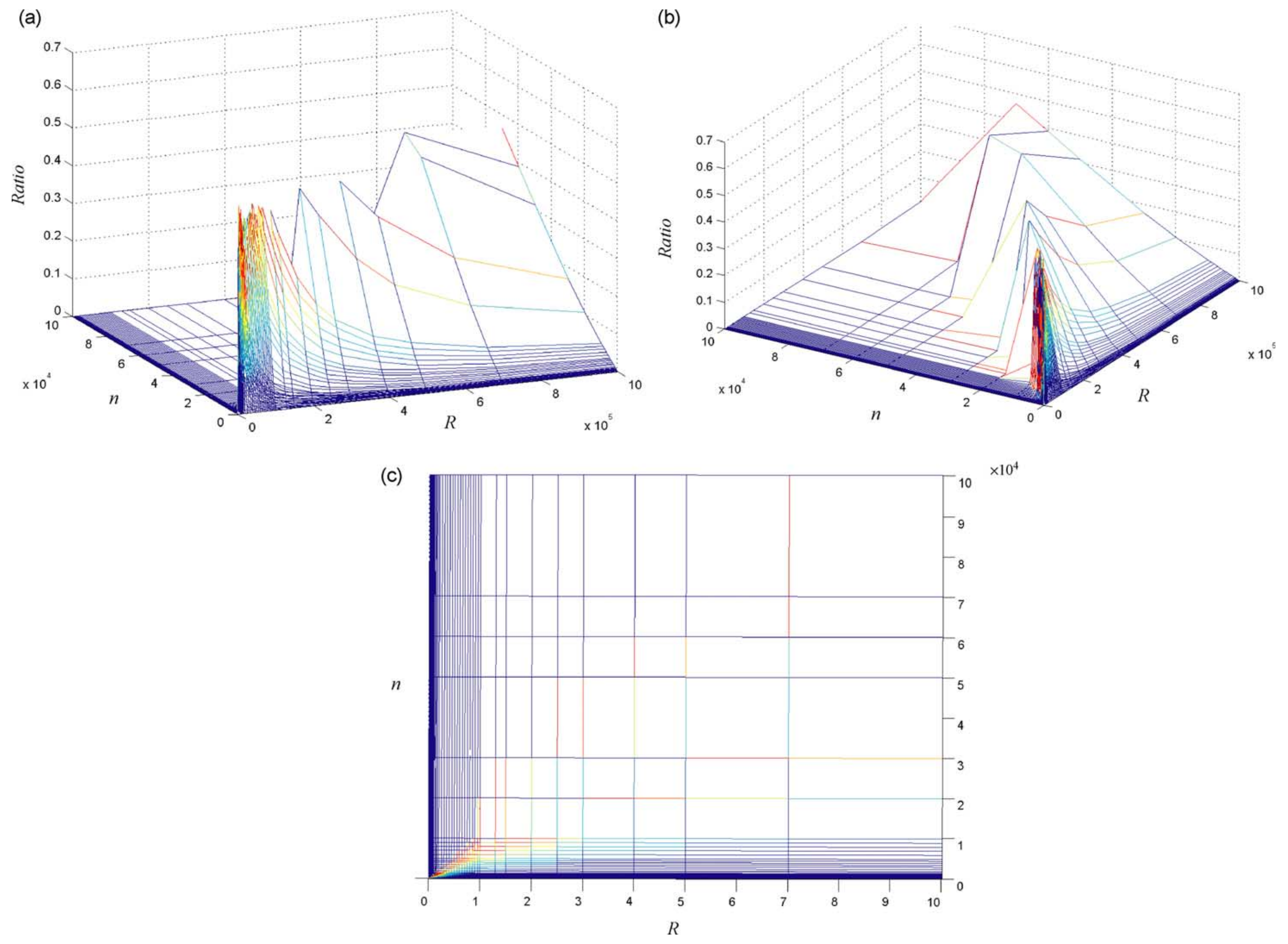


Fig. 5. (a) Plot of *Ratio* against *n*, *R*; (b) Plot of *Ratio* against *n*, *R*; and (c) Top view of *Ratio* under the combined effects of *n* and *R*.

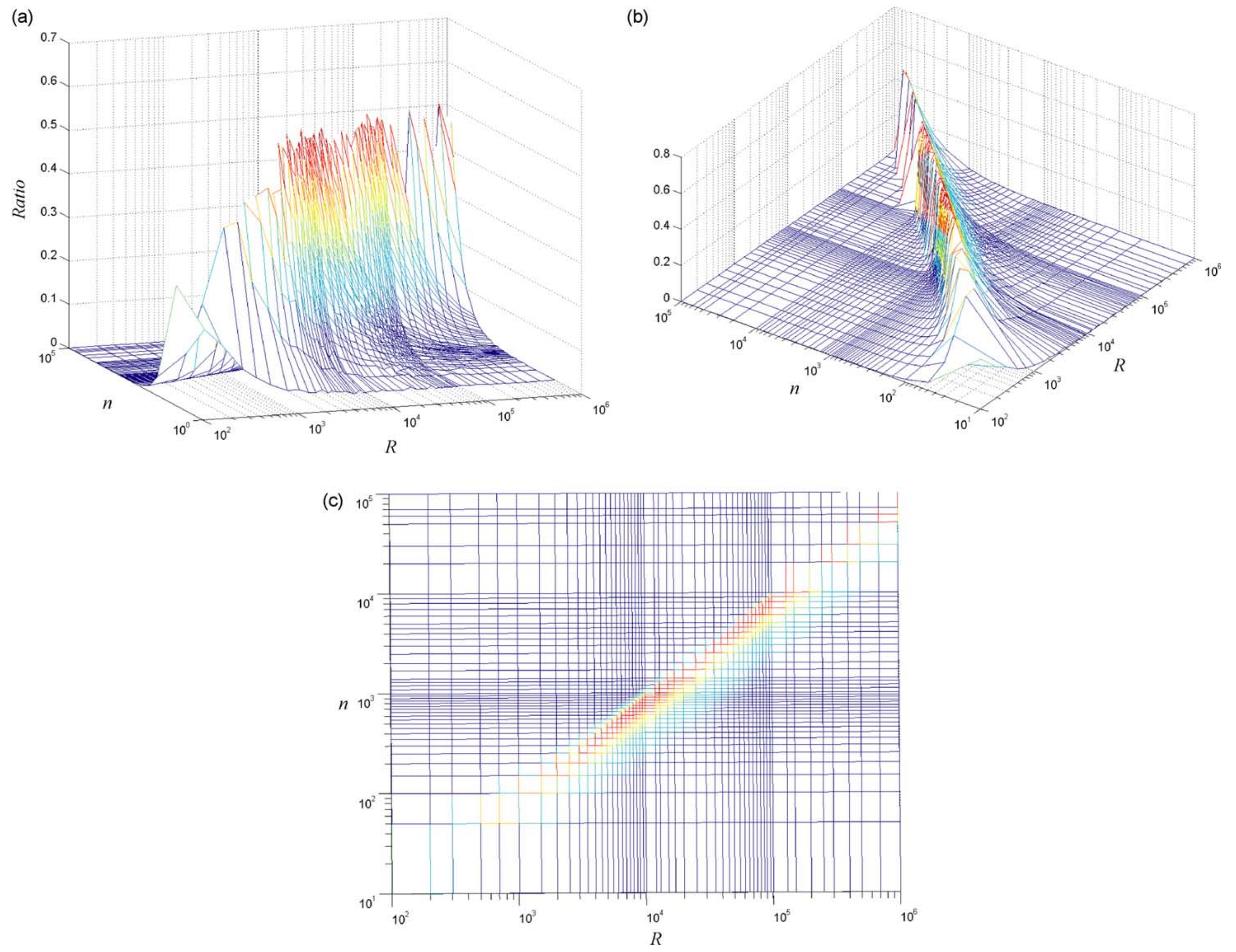


Fig. 6. (a) Plot of *Ratio* against *n*, *R*; (b) Plot of *Ratio* against *n*, *R*; and (c) Top view of *Ratio* under the combined effects of *n* and *R*.

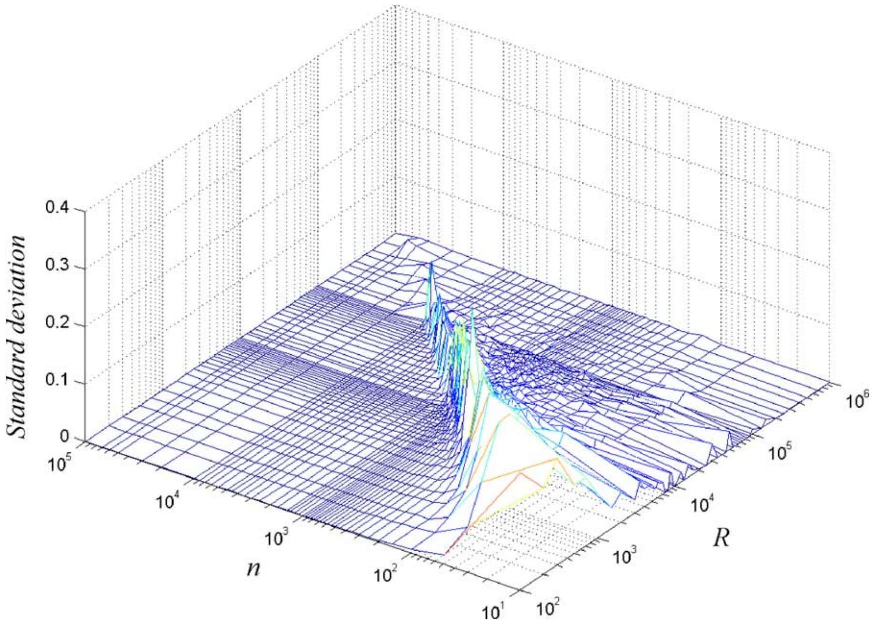


Fig. 7. Plot of standard deviation of test results against n, R.

5. Further comparisons

To further investigate the performance of the modified-PSW algorithm, we compared our results with the recent work on the single machine scheduling problem by Della Croce and T'kindt (2003) where an enhanced preemptive lower bound is derived from the well-known SRPT algorithm. This new lower bound is compared with the SRPT lower bound in Della Croce and T'kindt (2003) using test sets with up to 100 jobs, which were generated from Chu, 1992a,b with various parameter settings. The authors reported a gap average of 44% between the two lower bounds using the performance measure:

$$\text{(new lower bound} - \text{SRPT lower bound)/(optimal} - \text{SRPT lower bound)} \tag{3}$$

In this work, since we find exact solutions rather than lower bounds, we measured the quality of results using the *Ratio* performance measure defined in Section 2, which is a widely-accepted and standard method for measuring the performance of algorithms. We generated 900 test cases in all, following the same mechanisms given in Chu (1992a) and adopted the same set of parameters used in Della Croce and T'kindt (2003). For each job j, integer processing times were uniformly distributed within [1, 100] and the release times of jobs are integers uniformly distributed in the range $[1, n*50.5*p]$, where n is number of jobs and $n*50.5$ is the expected total processing time, and p is a range factor parameter which controls the closeness of job release times (Della Croce & T'kindt, 2003). We used the values 20, 40, 60, 80, 100 for n, and 0.2, 0.6, 1.0, 2.0, 3.0 for p, which is the same parameter range used in Della Croce and T'kindt (2003). For each (n, p) pair, 30 test sets were generated to give 900 test sets in total. Results are shown in Table 2.

Table 2

n	p	Avg. SRPT	Avg. MPSW	Avg. Gap (%)
20	0.20	5473.17	5936.53	8.85
20	0.60	3556.10	4111.57	18.24
20	1.00	2379.23	2865.93	20.15
20	1.50	1624.33	1871.30	15.28
20	2.00	1308.30	1439.27	9.99
20	3.00	1150.07	1234.17	6.94
40	0.20	21,952.37	22,929.00	4.51
40	0.60	12,765.67	14,118.87	11.72
40	1.00	5065.40	6135.33	21.79
40	1.50	3486.17	4037.20	15.59
40	2.00	2797.63	3068.50	9.45
40	3.00	2496.37	2668.10	7.05
60	0.20	46,378.40	48,080.50	3.68
60	0.60	26,433.83	28,875.37	9.87
60	1.00	9743.70	11,695.33	20.83
60	1.50	5383.33	6232.07	16.00
60	2.00	4207.40	4712.70	11.96
60	3.00	3697.77	3944.87	6.72
80	0.20	80,195.73	82,666.93	3.14
80	0.60	42,065.97	45,246.33	7.74
80	1.00	14,719.40	17,537.63	20.13
80	1.50	6995.47	8093.47	15.76
80	2.00	5528.23	6247.03	13.02
80	3.00	4867.00	5234.73	7.60
100	0.20	13,2480.23	13,5601.90	2.39
100	0.60	66,964.40	71,280.97	6.75
100	1.00	18,188.27	21,876.03	22.00
100	1.50	8985.73	10,457.63	16.29
100	2.00	6954.80	7719.23	11.08
100	3.00	5978.43	6383.27	6.73

From these results, we see that MPSW gives good results for test sets used in Della Croce and T'kindt (2003). This is because results obtained were very close to lower bounds with performance gaps between 3 and 20%. This implies that the modified-PSW provides solutions very close to the lower bounds. We note that the average 44% lower bound improvement reported in Della Croce and T'kindt (2003) is of limited value using formula (3) when the SRPT bound is already close to optimal solutions.

We note also that the range factor p had an impact on performance. It is clear that for different values of n, the hardest problems were those with the range factor p near 1.0 since for test sets with $p = 1.0$ the gap from the SRPT lower bound had values close to and above 20%. The difficulty of test sets decreased as p decreased or increased from the value 1.0. This is not surprising if we consider what setting $p = 1.0$ implies in test set generation. The release time is uniformly distributed in $[1, n*50.5]$ when $p = 1$, where $n*50.5$ is the expected total processing time. This means, jobs in these test sets are released from the start time to the expected finish time evenly across $[1, n*50.5]$ causing jobs to be tightly constrained. On the other hand, when p is smaller or larger than 1, jobs are released approximately at the same time or are

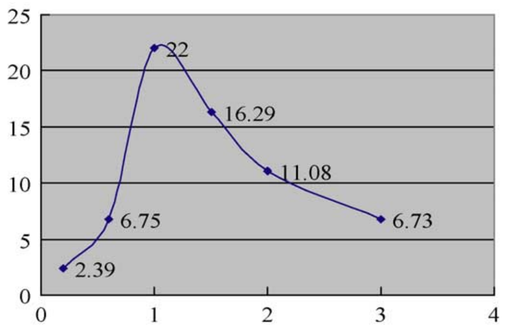


Fig. 8. Gap (%) against p when $n = 100$.

sufficiently separated. Both cases are easier to schedule since simple scheduling principles such as SRPT can be used. For these test sets, the gap is smaller as shown in Fig. 8.

6. Predictive model: development and extension

In this section, we develop a model to predict the average performance of the modified-PSW algorithm for the $1|r_j|\sum F_j$ problem. Although the above experiments are specific to the modified-PSW algorithm, our previous analysis is applicable to all algorithms that produce solution by converting preemptive schedules to non-preemptive ones, and therefore, the model developed can be used in predicting the performance of this type of algorithm. Testing the model provided satisfactory results even when some of the assumptions in this model (e.g. release time distribution) were violated.

6.1. Model development

In Section 4, we showed that every instance of $1|r_j|\sum F_j$ with factors $\{n, R, \mu\}$ can be transformed into an equivalent instance with factors $\{n, \alpha(R/\mu), \alpha\}$ for $\alpha > 0$ With this, we can predict the average performance of modified-PSW algorithm for cases with $\{n, R, \mu\}$ by finding the average performance for cases with $\{n, 10(R/\mu), 10\}$. The validity of prediction can be verified by graphically. For $\{n_0, R_0, 10\}$, we have

$$\log n_0 = k \log R_0 + b$$

After transforming $\{n, R, \mu\}$ to $\{n, 10(R/\mu), 10\}$, we substitute variables into above equation to get

$$\log n = k \log[10(R/\mu)] + b = k \log R + k \log(10/\mu) + b$$

Since k, b and μ are constants in each plot, we get a line with slope k and intercept $k \log(10/\mu + b)$. Hence, when we plot Fig. 6(a) for each μ value, ridge-projection lines will have the same slope k but different intercepts (see Fig. 9).

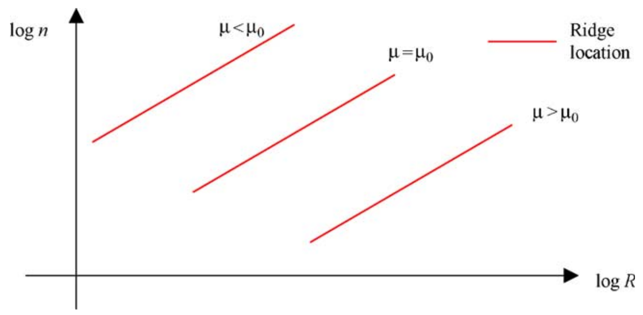


Fig. 9. Top view of relative ridge locations for different μ values.

6.2. Model testing

A performance prediction model for algorithms solving the $1|r_j|\sum F_j$ would have practical use for decision-makers. In this section, we test the accuracy of the above empirical prediction model using three types of problems: jobs with exponentially distributed processing time having $\mu=10$; jobs with exponentially distributed processing time with $\mu=100$; and jobs with uniformly distributed processing time with $\mu=10$. The release time follows a uniform distribution for all three cases. For each job type, there are 16 sampling points, each containing 50 test instances. With an interpolation function 'interp(n, R)', which approximates the *Ratio* value based on experimental data, for the first class of instances with factor $\{n, R, 10\}$, a simple application of interp(n, R) will give an approximate value. For the second class of problem instances, with factor $\{n, R, 100\}$, we use interp(n, R/100). The last class of instances with uniform processing time is studied for sensitivity testing to ascertain if a violation of model assumptions will affect accuracy. This is addressed by a direct application of interp(n, R). The interpolation function available in MATLAB ('interp2') can used in matrix form all the $44\times54=2376$ sampling points and their corresponding *Ratio* values. We randomly picked four different n and R values for each of the three kinds of jobs in testing the accuracy of the prediction model. In every table entry (except for table headers), the top value is the actual *Ratio* value obtained from running test cases, and the bottom value is the predicted *Ratio* value obtained from the empirical model. By comparing the actual *Ratio* from running test cases and the interpolated *Ratio* values, we find that

Table 3
Data for jobs with exponentially distributed processing time having $\mu=10$

n	*R*			
	826	8257	52,988	108,540
14	0.0939	0.0096	0.0005	0.0014
	0.0713	0.0049	0.0026	0.0014
343	0.0104	0.2446	0.0319	0.0139
	0.0116	0.2529	0.0293	0.0176
5468	0.0000	0.0005	0.0702	0.3152
	0.0000	0.0005	0.1872	0.3236
21,341	0	0.0000	0.0002	0.0008
	0	0	0.0002	0.0008

Table 4
Data for jobs with exponentially distributed processing time having $\mu=100$

n	R			
	826	8257	52,988	108,540
14	0.2665	0.0729	0.0058	0.0117
	N.A.	0.0714	0.0094	0.0030
343	0.0025	0.0138	0.4187	0.1883
	N.A.	0.0116	0.4141	0.1794
5468	0.0000	0.0001	0.0004	0.0008
	N.A.	0.0000	0.0003	0.0007
21,341	0	0	0.0000	0.0001
	N.A.	0	0	0.0000

All entries with N.A. indicate that the values to be predicted are not available because the (n, R/10) pairs are out of the range of the interpolation function in MATLAB.

the prediction value is acceptably accurate: the mean difference between the two *Ratio* values is 0.0069 for the first class, 0.0019 for the second class, and 0.0401, for the third class. Detailed test results are given in Tables 3–5.

7. Conclusion

In this study, we provide a modified-PSW algorithm from the $1|r_j|\sum F_j$ problem and conducted extensive experiments to determine the average performance of this algorithm. We also characterized the performance of the algorithm with respect to different combinations of parameters. It was found that performance was poor when the logarithms of n and R satisfy a linear relationship. We provided an analysis of this and gave empirical methods for judging average solution quality when the problem is solved by modified-PSW. In addition, we developed a model to predict the average performance for the general case. This model is easily used and is able to provide good predictions. We also compared our results for the problem with those given recently by Della Croce and T'kindt (2003).

Table 5
Data for jobs with uniformly distributed processing time having $\mu=10$

n	R			
	826	8257	52,988	108,540
14	0.0253	0.0017	0	0
	0.0713	0.0049	0.0026	0.0014
343	0.0084	0.0804	0.0085	0.0043
	0.0116	0.2529	0.0293	0.0176
5468	0.0001	0.0004	0.0267	0.1062
	0.0000	0.0005	0.1872	0.3236
21,341	0	0.0000	0.0002	0.0004
	0	0	0.0002	0.0008

References

Ahmadi, R. H., & Bagchi, U. (1990). Lower bounds for single-machine scheduling problems. *Naval Research Logistics*, *37*, 967–979.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Chandra, R. (1979). On $n|1|\bar{F}$ dynamic deterministic problems. *Naval Research Logistics*, *26*, 537–544.

Chu, C. (1992a). Efficient heuristics to minimize total flow time with release dates. *Operations Research Letters*, *12*, 321–330.

Chu, C. (1992b). A branch-and-bound algorithm to minimize total flow time with unequal release dates. *Naval Research Logistics*, *39*, 859–875.

Deogun, J. S. (1983). On scheduling with ready times to minimize mean flow time. *Computer Journal*, *26*, 320–328.

Della Croce, F., & T'kindt, V. (2003). Improving the preemptive bound for the one-machine dynamic total completion time scheduling problem. *Operations Research Letters*, *31*, 142–148.

Dessouky, M. I., & Deogun, J. S. (1981). Sequencing jobs with unequal ready times to minimize mean flow time. *SIAM Journal on Computing*, *19*, 192–202.

Hall, L. A., Shmoys, D. B., & Wein, J. (1996). Scheduling to minimize average completion time: offline and online algorithms, *Proceedings of the Seventh ACMSIAM Symposium on Discrete Algorithms,* pp. 142–151.

Jain, R. (1991). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New York: Wiley.

Kellerer, H., Tautenhahn, T., & Woeginger, G. J. (1996). Approximability and nonapproximability results for minimizing total flow time on a single machine, *Proceedings of the 28th Annual ACM Symposium on Theory of Computing,* pp. 418–426.

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

Leonardi, S., & Raz, D. (1997). Approximating total flow time on parallel machines, *ACM Symposium on Theory of Computing*.

Mao, W. Rifkin, A. *Analysis of the FCFS algorithm for* $1|r_j|\sum C_j$ unpublished manuscript.

Phillips, C., Stein, C., & Wein, J. (1995). Scheduling jobs that arrive over time, *Proceedings of the Fourth Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 955*. Berlin: Springer, pp. 86–97.

Phillips, C., Stein, C., & Wein, J. (1998). Minimizing average completion time in the presence of release dates. *Mathematical Programming B*, *82*, 199–224.

Pinedo, M. (1995). *Scheduling: theory, algorithms, and systems*. Englewood Cliff, NJ: Prentice Hall.

Smith, W. E. (1956). Various optimizers for single state production. *Naval Research Logistics Quarterly*, *3*, 56–66.