International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: https://www.tandfonline.com/loi/tprs20

Robust makespan minimisation in identical parallel machine scheduling problem with interval data

Xiaoqing Xu , Wentian Cui , Jun Lin & Yanjun Qian

To cite this article: Xiaoqing Xu , Wentian Cui , Jun Lin & Yanjun Qian (2013) Robust makespan minimisation in identical parallel machine scheduling problem with interval data, International Journal of Production Research, 51:12, 3532-3548, DOI: 10.1080/00207543.2012.751510

To link to this article: https://doi.org/10.1080/00207543.2012.751510

International Journal of Production Research, 2013
Vol. 51, No. 12, 3532–3548, http://dx.doi.org/10.1080/00207543.2012.751510

Taylor & Francis
Taylor & Francis Group

Robust makespan minimisation in identical parallel machine scheduling problem with interval data

Xiaoqing Xu[a], Wentian Cui[a], Jun Lin[a]* and Yanjun Qian[b]

[a]*School of Management, Xi'an Jiaotong University, Xi'an 710049, PR China;* [b]*School of Public Policy and Administration, Xi'an Jiaotong University, Xi'an 710049, PR China*

Parallel machine scheduling problems are commonly encountered in a wide variety of manufacturing environments and have been extensively studied. This paper addresses a makespan minimisation scheduling problem on identical parallel machines, in which the specific processing time of each job is uncertain, and its probability distribution is unknown because of limited information. In this case, the deterministic or stochastic scheduling model may be unsuitable. We propose a robust (min–max regret) scheduling model for identifying a robust schedule with minimal maximal deviation from the corresponding optimal schedule across all possible job-processing times (called scenarios). These scenarios are specified as closed intervals. To solve the robust scheduling problem, which is NP-hard, we first prove that a regret-maximising scenario for any schedule belongs to a finite set of extreme point scenarios. We then derive two exact algorithms to optimise this problem using a general iterative relaxation procedure. Moreover, a good initial solution (optimal schedule under a mid-point scenario) for the aforementioned algorithms is discussed. Several heuristics are developed to solve large-scale problems. Finally, computational experiments are conducted to evaluate the performance of the proposed methods.

Keywords: makespan minimisation; identical parallel machines; min–max regret; uncertain processing times

1. Introduction

Recently, many variants of parallel machine scheduling problems have been studied because of their theoretical and practical importance (see, e.g., Chang et al. 2008; Chen 2008; Chang Chung, Tai, and Pearn 2009; Jia and Mason 2009; Mason, Jin, and Jampani 2009; Ozlen and Azizoglu 2009; Chuang, Liao, and Chao 2010; Hu, Bao, and Jin 2010; Ozlen and Webster 2010; Shin and Kang 2010; Gamberini et al. 2011; Kaczmarczyk 2011). These problems span a wide variety of manufacturing environments, such as the semiconductor manufacturing industry (Chien and Chen 2007), the manufacture of thin-film transistor liquid crystal displays (TFT-LCD) (Chung, Tai, and Pearn 2009), the manufacturing of anodic electro-etching aluminium foil (Chuang, Liao, and Chao 2010), and the electronics industry (Kaczmarczyk 2011). Most of the literature considers problems in environments where the processing times of jobs are assumed to be deterministic and precisely known in advance.

In practice, however, the exact processing times are unknown, and the values often vary because of the uncertainties presented by a few factors, such as machine or tool conditions, worker skill levels, and production environments (Daniels and Kouvelis 1995; Yang and Yu 2002; Montemanni 2007; Allahverdi and Aydilek 2010). For these reasons, the optimal schedule obtained by the deterministic model with one forecast instance may deviate considerably from actual optimal schedules. Several approaches should be used to handle parallel machine scheduling problems with uncertain processing times.

Stochastic scheduling models have been developed for the parallel machine scheduling problem with uncertain processing times (see, e.g., Vanderheyden 1981; Weber 1982). In the stochastic approach, uncertain processing times are modelled by specifying the probability distributions on the basis of a substantial volume of historical data. Unfortunately, historical data are not always available to estimate the probability distributions (e.g., factories often diversify their products or introduce new product typologies in production plans) (Anglani et al. 2005; Allahverdi and Aydilek 2010). Moreover, risk-averse decision-makers are more interested in hedging against worst-case performance than in optimising expected performance, especially when a job set is encountered only once (Daniels and Kouvelis 1995; Kouvelis and Yu 1997; Yang and Yu 2002; Lin and Ng 2011). For example, for the new product-development division of ZTE corporation in China, there are several prototyping facilities through which dozens of mobile-phone develop-

*Corresponding author. Email: linjun@alumni.nus.edu.sg; ljun@mail.xjtu.edu.cn

ment projects have to be scheduled. Prototype building of each new mobile phone, which might also involve a complex interface of simultaneous product and process technology development, often results in tremendous uncertainty in terms of duration of the development task. Because the set of products is encountered for the first time, historical data are not available. In such cases, stochastic models are inapplicable. The robust scheduling of the development projects of mobile phone through the crucial prototype building stage is a viable way to hedge against the inherent uncertainty.

The book written by Kouvelis and Yu (1997) provides a general framework and a wide review of robust optimisation models. In the robust approach, the processing times of jobs can be structured through scenarios, each corresponding to a potential realisation of uncertain processing times. Two methods are typically used to describe the set of all possible scenarios: interval and discrete scenarios.

In the interval scenario case, the processing time of each job can take on any value from a range given lower and upper bounds. In the discrete scenario case, the scenario set is described explicitly. In the current paper, we define uncertain processing times as intervals because in many cases, all that is available is an educated guess of the lower and upper bounds of processing times (Allahverdi and Aydilek 2010). Three robustness criteria are used: absolute robustness, robust deviation, and relative deviation. We apply the robust deviation criterion, also called min–max regret. The min–max regret approach, which stems from decision theory, aims to identify a robust solution that minimises the maximal deviation of a given solution from the optimum across all possible scenarios. A decision using this criterion is appropriate in highly competitive environments, where the performance of the corporation is required to be close to that of its competitors in any set of potential realizable scenarios, such as aforementioned the mobile phone market (Kouvelis and Yu 1997). This is because the min–max regret criterion bounds the magnitude of missed opportunities by seeking a decision which leads to a performance close to that of the optimal decisions in any scenarios.

The min–max regret approach has been applied to many combinatorial optimisation problems with interval input, such as the shortest path (Montemanni and Gambardella 2004, 2005; Aissi, Bazgan, and Vanderpooten 2005a), spanning tree (Aissi, Bazgan, and Vanderpooten 2005a; Montemanni and Gambardella 2004, 2005; Kasperski and Zielinski 2006; Monternanni 2006), aassignment (Aissi, Bazgan, and Vanderpooten 2005b), and knapsack (Aissi, Bazgan, and Vanderpooten 2005a; Deineko and Woeginger 2010). Aissi, Bazgan, and Vanderpooten (2009) reviewed the min–max regret approach, which has also been applied to production scheduling problems with interval processing times. Daniels and Kouvelis (1995) studied min–max regret scheduling problems with total flow time criterion on a single machine. Kouvelis, Daniels, and Vairaktarakis (2000) discussed min–max regret makespan minimisation in a two-machine flow shop. These two problems are NP-hard in the interval scenario case; heuristics and branch-and-bound algorithms have been developed to solve them (Daniels and Kouvelis 1995; Kouvelis, Daniels, and Vairaktarakis 2000). The min–max regret problem of minimising the total flow time on a single machine, discussed by Daniels and Kouvelis (1995), has been investigated further by other scholars. Lebedev and Averbakh (2006) showed that the processing time intervals of jobs with the same centre can be addressed in $O(n\log n)$time if the number of jobs is even; this problem is NP-hard if the number of jobs is odd. Montemanni (2007) proposed a mixed-integer linear programming formulation of this problem and showed the manner in which preprocessing rules are transformed into valid inequalities. Kasperski and Zielinski (2008) proved that the optimal schedule under the mid-point scenario guarantees a 2-approximation of the optimal solution. Lu, Lin, and Ying (2011) discussed a case in which uncertain job-processing times and sequence-dependent family setup times are both explicitly represented by interval data. The authors reformulated this problem as a robust constrained shortest path problem, and solved it using an SA-based heuristic. A min–max regret scheduling problem with maximum lateness criterion and interval processing times of jobs on a single machine was discussed by Kasperski (2005). The author also developed an $O(n^4)$ algorithm. In Averbakh (2006), the min–max regret makespan-minimising permutation flow shop problem with two jobs, m machines, and interval job-processing times was considered. An $O(m)$ algorithm was also presented. All of the abovementioned studies focus on a single machine-scheduling problem with interval data. Parallel machine-scheduling problems with uncertain processing times whose exact probability distributions are unknown have been rarely investigated. To the best of our knowledge, only Anglani et al. (2005) have studied the problem of scheduling independent lots on identical parallel machines with sequence-dependent setup costs under the hypothesis of fuzzy processing times. The objective of this research was to minimise the total setup cost.

In the current work, we investigate the makespan minimisation problem on identical parallel machines, in which we assume that the processing times are uncertain but lie in intervals. To solve this problem, we first propose a concept of critical machine inspired by the concept of critical path and job in Kouvelis, Daniels, and Vairaktarakis (2000); Kasperski (2005), and Conde (2009); we then prove two properties of worst-case (regret-maximising) scenarios and the maximal regret for a given schedule using this concept as a basis. The aforementioned procedures are presented in Section 2. In Section 3, we present two exact algorithms to identify the optimal solution on the basis of an iterative relaxation algorithm proposed by Inuiguchi and Sakawa (1995) and Mausser and Laguna (1998, 1999). Moreover, a good initial solution, i.e. the optimal schedule of the mid-point scenario, is discussed. In Section 4, heuristic algorithms,

such as the local search algorithm and simulation annealing algorithm, are developed to solve large-scale problems. In Section 5, we present the computational experiments to evaluate the performance of the proposed methods. Finally, Section 6 summarises our findings and directions for further work.

2. Formulation of a robust identical parallel machine scheduling problem

Let J={1,2,...,n} be the set of n independent and non-preemptive jobs to be scheduled, and M={1,2,...,m} denote the set of m identical parallel machines. Each job is available at time 0 and can be processed only by one arbitrary machine. No machine can process more than one job at the same time.

We are interested in a scheduling environment where the processing times of jobs are uncertain, and their probability distribution is unavailable because of limited information. The processing time of each job $i \in J$ is assumed to lie within a closed interval $[\underline{p_i}, \bar{p}_i]$, where $0 \le \underline{p_i} \le \bar{p}_i$. A scenario $\boldsymbol{s} = \{p_1^s, p_2^s, \ldots, p_n^s\}$ is a possible realisation of the processing times of jobs, and $p_i^s \in [\underline{p_i}, \bar{p}_i], i \in J$ denotes the processing time of job i under scenario $\boldsymbol{s}$. Then, the processing time uncertainty can be described through S, the set of all the scenarios. A schedule can be represented by a matrix $\boldsymbol{X} = [x_{ij}]_{n\times m}$, so that if job $i \in J$ is assigned to the machine, $j \in M, x_{ij}$ equals 1; otherwise, x_{ij} equals 0. A feasible schedule should satisfy $\sum_{j=1}^{m} x_{ij} = 1, i = 1, 2, \ldots, n$, because each job is to be processed on exactly one machine. Let Φ be the set of all feasible schedules. The completion time of machine j in schedule $\boldsymbol{X}$ under scenario $\boldsymbol{s}$ is computed as follows:

$$C_j^s(\boldsymbol{X}) = \sum_{i=1}^{n} p_i^s x_{ij}. \tag{1}$$

The makespan of schedule $\boldsymbol{X}$ under scenario $\boldsymbol{s}$ is defined as:

$$F(\boldsymbol{X}, \boldsymbol{s}) = \max_{j\in M}\{C_j^s(\boldsymbol{X})\} \tag{2}$$

We denote the minimum makespan under scenario $\boldsymbol{s}$ as F_s^*, and the corresponding optimal schedule as $\boldsymbol{X}_s^*$:

$$F_s^* = F(\boldsymbol{X}_s^*, \boldsymbol{s}) = \min_{X\in\Phi} F(\boldsymbol{X}, \boldsymbol{s}). \tag{3}$$

Seeking a schedule whose makespan is minimal under a fixed scenario is the same as optimising a deterministic $P\|C_{max}$ scheduling problem. Given a feasible schedule $\boldsymbol{X} \in \Phi$, its regret under scenario $\boldsymbol{s}$ is defined as:

$$R(\boldsymbol{X}, \boldsymbol{s}) = F(\boldsymbol{X}, \boldsymbol{s}) - F_s^*. \tag{4}$$

The maximum regret of schedule $\boldsymbol{X}$ is then defined as:

$$R_{max}(\boldsymbol{X}) = \max_{\boldsymbol{s}\in S} R(\boldsymbol{X}, \boldsymbol{s}). \tag{5}$$

The scenario that maximises the regret across all possible scenarios is called the worst-case scenario for $\boldsymbol{X}$. The robust parallel machine scheduling problem (RPMS) can be formulated as:

$$\min_{\boldsymbol{X}\in\Phi} R_{max}(\boldsymbol{X}) = \min_{\boldsymbol{X}\in\Phi} \max_{\boldsymbol{s}\in S}(F(\boldsymbol{X}, \boldsymbol{s}) - F_s^*). \tag{6}$$

The solution corresponds to the optimal robust schedule.

The RPMS problem can be viewed as a generalisation of the classical $P\|C_{max}$ scheduling problem. If $\underline{p_i} = \bar{p}_i$ for all $i \in J$, then only one scenario $\boldsymbol{s}$ exists, and Equation (6) is equivalent to the problem of identifying an optimal schedule with the minimal makespan under scenario $\boldsymbol{s}$. Given that $P\|C_{max}$ is NP-hard, the general RPMS problem is also NP-hard.

The RPMS problem can also be formulated as the following mathematical model:

$$(RPMS)\ \min_{\boldsymbol{X}}\{\max_{\boldsymbol{s}\in S}[F(\boldsymbol{X}, \boldsymbol{s}) - F_s^*]\}, \tag{7}$$

$$s.t. \quad \sum_{j=1}^{m} x_{ij} = 1,\ i = 1, 2, \ldots, n, \tag{8}$$

$$x_{ij} \in \{0, 1\},\ i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m, \tag{9}$$

where F_s^* can be determined by solving a mixed-integer linear programming problem. This formulation is nonlinear because of the two nested max operators (note: $F(\boldsymbol{X}, \boldsymbol{s}) = max_{j \in M}\{\sum_{i=1}^{n} p_i^s x_{ij}\}$) in the objective function. However, we can transform it into a mixed-integer linear programming model:

$$(RPMS) min\, r$$

$$s.t. \sum_{i=1}^{n} p_i^s x_{ij} - F_s^* \leq r,\ \boldsymbol{s} \in S; j = 1, 2, \ldots, m, \tag{10}$$

$$(8) \text{ and } (9)$$

This mixed-integer linear programming model contains an infinite number of constraints and cannot be directly solved because $|S| = \infty$. Therefore, we develop two properties to characterise the worst-case scenarios. With these results, attention can be restricted to a finite set of discrete scenarios.

To describe succinctly and clearly the properties of the worst-case scenarios and maximum regret, two definitions are given:

DEFINITION 1. A machine $f \in M$ is said to be critical in a schedule $\boldsymbol{X} \in \Phi$ under scenario $\boldsymbol{s} \in S$, if f is the machine with maximum completion time in $\boldsymbol{X}$ under $\boldsymbol{s}$; i.e.,

$$C_f^s(\boldsymbol{X}) = \max_{j \in M}\{C_j^s(\boldsymbol{X})\} = F(\boldsymbol{X}, s).$$

DEFINITION 2. An extreme point scenario $\boldsymbol{s}^j, j \in M$, for schedule $\boldsymbol{X}$ is defined as follows:

$$p_i^{s^j} = \begin{cases} \bar{p}_i, \text{ if } x_{ij} = 1, \\ \underline{p}_i, \text{ if } x_{ij} = 0, \end{cases} \quad i = 1, 2, \ldots, n$$

or

$$p_i^{s^j} = \bar{p}_i x_{ij} + \underline{p}_i (1 - x_{ij}),\ i = 1, 2, \ldots, n$$

Property 1. For any schedule $\boldsymbol{X} \in \Phi$, let $\boldsymbol{s}^0$ be a worst-case scenario for $\boldsymbol{X}$ in which machine $f \in M$ is critical. Then, a scenario $\boldsymbol{s}^f$ exists for schedule $\boldsymbol{X}$ such that

(a) Machine f is also critical in $\boldsymbol{X}$ under $\boldsymbol{s}^f$, and
(b) Scenario $\boldsymbol{s}^f$ is a worst-case scenario for $\boldsymbol{X}$.

(All proofs are provided in Appendix 1.)

Consider a feasible schedule $\boldsymbol{X}$ with worst-case scenario $\boldsymbol{s}^f$, which satisfies Conditions a–b of Property 1. Because f is critical in $\boldsymbol{X}$ under $\boldsymbol{s}^f$, we have $F(\boldsymbol{X}, \boldsymbol{s}^f) = \sum_{i=1}^{n} \bar{p}_i x_{if}$ and obtain the maximum regret for $\boldsymbol{X}$:

$$R_{max}(\boldsymbol{X}) = F(\boldsymbol{X}, \boldsymbol{s}^f) - F_{\boldsymbol{s}^f}^* = \sum_{i=1}^{n} \bar{p}_i x_{if} - F_{\boldsymbol{s}^f}^*. \tag{11}$$

However, Property 1 cannot be applied directly because we do not know which machine is critical. Therefore, the following property is proposed to help us identify the critical machine and maximum regret.

Property 2. Given a schedule $\boldsymbol{X}$, the maximal regret for this schedule can be expressed as follows:

$$R_{max}(\boldsymbol{X}) = \max_{j \in M}\left\{\sum_{i=1}^{n} \bar{p}_i x_{ij} - F_{\boldsymbol{s}^j}^*\right\}. \tag{12}$$

Using Equations (11) and (12), we can also identify a worst-case scenario $\mathbf{s}^{j^*}$ for $\boldsymbol{X}$, in which machine j^* is critical, where $j^* = arg\,max_{j\in M}\left\{\sum_{i=1}^{n}\bar{p}_i x_{ij} - F^*_{s^j}\right\}$. Replacing Equation (10) with Equation (12) enables the reformulation of the RPMS problem as follows:

$$\begin{gathered}\min r \\ s.t. \sum_{i=1}^{n}\bar{p}_i x_{ij} - F^*_{s^j} \le r, \quad j = 1,2,\ldots,m, \\ (8) \text{ and } (9)\end{gathered} \tag{13}$$

where $F^*_{s^j}$ is the minimal makespan under scenario $\boldsymbol{s}^j$. Let matrix $\boldsymbol{Y} = [y_{ik}]_{n\times m}$ denote the corresponding optimal schedule under scenario $\boldsymbol{s}^j$. Variables y_{ik} are such that $y_{ik} = 1$ if job i is on machine k in the optimal schedule; otherwise, $y_{ik} = 0$, and the variables satisfy $\sum_{k=1}^{m} y_{ik} = 1, i = 1,2,\ldots,n$. Then,

$$F^*_{s^j} = \max_{k\in M}\left\{\sum_{i=1}^{n} p_i^{s^j} y_{ik}\right\} = \max_{k\in M}\left\{\sum_{i=1}^{n}(\bar{p}_i x_{ij} + \underline{p}_i(1 - x_{ij}))y_{ik}\right\}. \tag{14}$$

Because of the interaction between the $\boldsymbol{X}$ and $\boldsymbol{Y}$ variables in Equation (14), inequality (13) is nonlinear. Therefore, the RPMS problem cannot be solved directly using mixed-integer linear programming. Iterative or heuristic methods are needed.

3. Exact algorithms for the RPMS problem

RPMS is a min–max problem that can be solved using a general iterative relaxation (IR) procedure proposed in Inuiguchi and Sakawa (1995) and Mausser and Laguna (1998, 1999). First, the set of all possible scenarios S is replaced in RPMS with a finite set of scenarios $\Gamma = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_h\}$, resulting in a relaxed mixed-integer program (RPMS-relaxed):

$$min\ r$$

$$s.t. \sum_{i=1}^{n} p_i^{s_k} x_{ij} - F^*_{s_k} \le r, \qquad \forall s_k \in \Gamma; j = 1,2,\ldots,m,$$

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad i = 1,2,\ldots,n,$$

$$x_{ij} \in \{0,1\}, \qquad i = 1,2,\ldots,n; j = 1,2,\ldots,m,$$

The minimal makespan $F^*_{s_k}$ can always be obtained by solving a mixed-integer programming if we know scenario $s_k, 1 \le k \le h$. We refer to the m constraints $\sum_{i=1}^{n} p_i^{s_k} x_{ij} - r \le F^*_{s_k}, j = 1,2,\ldots,m.$, associated with $\forall \boldsymbol{s}_k \in \Gamma$, as regret cuts. Let $\hat{\boldsymbol{X}}$ denote the solution to RPMS-relaxed, with corresponding objective value $\hat{r}$. Let us assume that the minimal maximum regret for RPMS is r^*. Note that $\hat{r} \le r^*$ is a lower bound for the RPMS problem, and this lower bound is non-decreasing as more regret cuts are added to RPMS-relaxed.

Given a candidate solution $\hat{\boldsymbol{X}}$, we can obtain its worst-case scenario $\hat{\boldsymbol{s}}$ and maximal regret $R_{max}(\hat{\boldsymbol{X}})$ on the basis of Properties 1 and 2. Note that $R_{max}(\hat{\boldsymbol{X}}) \ge r^*$ is an upper bound for the RPMS problem. If the upper bound is greater than the current lower bound, the following iterative procedure continues. New scenario $\hat{\boldsymbol{s}}$ is added into original Γ to construct new Γ. Then, the new regret cuts for RPMS-relaxed are generated. In solving RPMS-relaxed, we obtain a relaxed solution and an objective value, which are also the candidate solution and lower bound for the next iteration. If the upper bound is lower than or equal to the current lower bound, this iterative procedure stops, and $\hat{\boldsymbol{X}}$ becomes the robust optimal solution.

In this IR procedure, RPMS-relaxed becomes more difficult to solve because the number of regret cuts increases with the number of iterations. Moreover, to obtain the worst-case scenario and maximal regret at each iteration, we have to solve the NP-hard problem $P\|C_{max}\,m$ times (see Property 2), which is a time-consuming process. The above-mentioned discussion clearly shows that two potential approaches to improving the IR algorithm are available: reducing the number of iterations or the number of $P\|C_{max}$ problems to be solved at each iteration.

3.1 *Iterative relaxation procedure with tight upper bound (IR-U)*

The IR procedure ensures that lower bound LB is non-decreasing as the iterative procedure continues. However, this does not guarantee that upper bound UB is non-increasing. The IR procedure stops when $LB = UB$, where UB is the maximum regret for current relaxed solution $\hat{\boldsymbol{X}}$. If, at each iteration, we record the minimal maximum regret obtained as far as current upper bound UB^* and its corresponding solution $\boldsymbol{X}^*$ are concerned, the algorithm stops when $LB = UB^* \leq UB$. The optimal robust solution is $\boldsymbol{X}^*$ rather than $\hat{\boldsymbol{X}}$. UB^* is non-increasing, and the number of iterations may be reduced.

Procedure of IR-U

Step 0: Set $LB = 0$, $UB^* = +\infty$ and choose an initial solution $\hat{\boldsymbol{X}}$.

Step 1: Worst-case scenario $\hat{\boldsymbol{s}}$ is identified, and maximum regret $R_{max}(\hat{\boldsymbol{X}})$ is calculated on the basis of Properties 1 and 2. If $UB^* \geq R_{max}(\hat{\boldsymbol{X}})$, set $UB^* = R_{max}(\hat{\boldsymbol{X}})$ and $\boldsymbol{X}^* = \hat{X}$. If $UB^* \leq LB$, go to Step 4.

Step 2: Add regret cuts $\sum_{i=1}^{n} p_i^{\hat{s}} x_{ij} - r \leq F_{\hat{s}}^*, j = 1, 2, \ldots, m$, to RPMS-relaxed.

Step 3: $\hat{\boldsymbol{X}}$ and $\hat{r}$ are identified by solving RPMS-relaxed. Set $LB = \hat{r}$ and go to Step 1.

Step 4: STOP.

The efficiency of this algorithm depends considerably on the choice of a suitable initial solution. An initial solution can be generated by solving a deterministic $P\|C_{max}$ problem under the mid-point scenario, i.e. $(\underline{p}_i + \bar{p}_i)/2, \forall i \in J$. A good initial solution should have a low maximum regret. The following section describes the initial solution and derives an upper bound on its maximum regret.

Let α_i denote the ratio of the length of the interval to the lower bound of the processing time of job i, and α be the maximum value among all $\alpha_i, i \in j$, i.e. $\alpha = max_{i \in J}\{\alpha_i\}$, where $\alpha_i = (\bar{p}_i - \underline{p}_i)/\underline{p}_i$. Let $\boldsymbol{s}^{\frac{1}{2}}$ be the mid-point scenario, i.e., $p_i^{s^{\frac{1}{2}}} = (\bar{p}_i + \underline{p}_i)/2, i \in J$, and $\boldsymbol{X}$ an optimal schedule under $\boldsymbol{s}^{\frac{1}{2}}$, i.e., $F\left(\boldsymbol{X}, s^{\frac{1}{2}}\right) = F_{s^{\frac{1}{2}}}^*$.

Property 3. Let us assume that $\boldsymbol{X}$ is an optimal schedule under the mid-point scenario. Then we derive an upper bound on the maximum regret of $\boldsymbol{X}$, $\frac{2\alpha}{2+\alpha} F_{s^{\frac{1}{2}}}^*$.

Property 3 indicates that if we choose $\boldsymbol{X}$ as our solution to the RPMS problem, the regret cannot be greater than $\frac{2\alpha}{2+\alpha} F_{s^{\frac{1}{2}}}^*$. This result is important in enabling decision-makers to directly use the optimal schedule under a mid-point scenario and avoid the time-consuming procedure of identifying the optimal schedule when this upper bound is acceptable. Otherwise, the optimal schedule under the mid-point scenario can be used as the initial schedule of the exact algorithms and heuristics.

3.2 *Iterative relaxation procedure with the worse-case scenario (IR-S)*

In the IR procedure, adding worst-case scenario $\hat{\boldsymbol{s}}$ for a relaxed solution $\hat{\boldsymbol{X}}$ into Γ at each iteration is not an absolute necessity. Any scenario in which the regret for $\hat{\boldsymbol{X}}$ exceeds the current lower bound can be added into Γ. Therefore, instead of identifying a worst-case scenario among m scenarios $\boldsymbol{s}^j, j \in \mathrm{M}$ and calculating the maximal regret for $\hat{\boldsymbol{X}}$ on the basis of Properties 1 and 2, we can stop examining the other scenarios as long as we find a worse-case scenario $\boldsymbol{s}^k$, under which the regret is greater than the current lower bound. This scenario can be added into Γ. With this method, we can reduce the number of times of calculating $P\|C_{max}$ if $k<m$. If, for a schedule $\hat{\boldsymbol{X}}$, no scenario leads to a regret greater than the current lower bound, then this schedule is optimal.

Procedure of IR-S

Step 0: Set $LB = 0$ and choose the optimal schedule under the mid-point scenario as initial solution $\widehat{\boldsymbol{X}}$.

Step 1: Set $k = 0$.

Step 2: Set $k = k + 1$. If $k>m$, go to Step 5. If $k \leq m$, calculate $R(\widehat{\boldsymbol{X}}, \boldsymbol{s}^k)$. If $R(\widehat{\boldsymbol{X}}, \boldsymbol{s}^k)>LB$, set $\hat{\boldsymbol{s}} = \boldsymbol{s}^k$; Otherwise, repeat Step 2.

Step 3: Add m cuts $\sum_{i=1}^{n} p_i^{\hat{s}} x_{ij} - r \leq F_{\hat{s}}^*, j = 1, 2, \ldots, m$, to RPMS-relaxed.

Step 4: $\hat{\boldsymbol{X}}$ and $\hat{r}$ are identified by solving RPMS-relaxed. Set $LB = \hat{r}$ and go to Step 1.

Step 5: STOP.

Although the IR-S algorithm reduces the number of scenarios examined at each iteration, it does not dominate the IR-U algorithm, which decreases the total number of iterations using the information on the lower and upper bounds. The efficiency of these two algorithms is examined in Section 5.

4. Heuristics for the robust scheduling problem

The proposed exact algorithms are computationally expensive because RPMS-relaxed is a mixed-integer linear programming problem, whose difficulty increases with the number of regret cuts added into it. Therefore, efficient heuristics are needed for solving large-scale problems.

A schedule can be changed into a new one by moving a job from one machine to another, i.e. *shift neighbourhood*, or interchanging two jobs on different machines, i.e. *interchange neighbourhood*. If the new schedule has a lower maximum regret, we say that the original schedule is improved. If no new schedule with a lower maximum regret can be identified, the current schedule is locally optimal. The issue now becomes: how do we move or interchange jobs? The following properties provide some clues.

Property 4. If a given schedule can be improved by moving a job from one machine to another, the job must be on the critical machine under the worst-case scenario.

Property 5. If a given schedule can be improved by interchanging two jobs on different machines, one of the two jobs must be on the critical machine under the worst-case scenario.

These properties are used in the following heuristic algorithms to generate new schedules and reduce the maximum regret step by step.

4.1 *Hill-climbing local search algorithm*

The hill-climbing local search (LS) algorithm is a general approach to designing heuristics for optimisation problems. A brief description of the procedure is as follows. Given an initial schedule, neighbour schedules are generated. If a better neighbour schedule is found, the current schedule is replaced with the better schedule, and the same procedure is repeated. If none of the neighbour schedules are better than the current schedule, the procedure is terminated. Three important elements of the LS algorithm in this study are described as follows:

(a) Initial schedule. The optimal schedule under the mid-point scenario is chosen.
(b) Neighbourhood generation. We use two methods to generate the neighbours of the incumbent schedule: *shift* and *interchange* neighbourhoods, which are also adopted by Józefowska et al. (1998) and Kouvelis, Daniels, and Vairaktarakis (2000). Using the shift neighbourhood, we can generate $(m-1)n$ alternative schedules. Conversely, $C_n^2 = (n-1)n/2$ alternative schedules can be generated using the interchange neighbourhood because $C_n^2 = (n-1)n/2$ potential pairwise interchanges can be incorporated into the incumbent schedule. Only the $(m-1)n_c$ shift schedules and $(n-n_c)n_c$ interchange schedules that do not violate Properties 4 and 5 are evaluated, where n_c is the number of jobs on the critical machine under the worst-case scenario. The two neighbourhood generation mechanisms are examined using the same method described in Józefowska et al. (1998). The results indicate that the interchange neighbourhood yields better results than the shift neighbourhood for all the 750 test problems (the method for generating the test problems is illustrated in Section 5). Thus, the interchange neighbourhood is applied in this paper.
(c) Stop criterion. If none of the alternative schedules are better than the current schedule, the procedure is terminated.

Procedure of LS

Step 0: Compute the optimal schedule $\boldsymbol{X}$ under the mid-point scenario and use it as the initial schedule. Determine the maximum regret and the critical machine for schedule $\boldsymbol{X}$ under its worst-case scenario using Property 2. Let $R_{max}(\boldsymbol{X})$ and cm be the maximum regret and critical machine, respectively. If $R_{max}(\boldsymbol{X}) = 0$, go to Step 3. Set $k = 0, N = 0$.

Step 1: Set $k = k + 1$. If $k = n$, then $k = 1$.

Step 2: Set $j = k + 1, N = N + 1$. If $j>n$, go to Step 1. If jobs j and k are on the same machine, or neither of them is on the critical machine cm, repeat Step 2. Otherwise, construct a new schedule $\boldsymbol{X}'$ by interchanging jobs j and k in schedule $\boldsymbol{X}$. Determine the maximum regret and the critical machine for $\boldsymbol{X}'$ under its worst-case scenario and let $R_{max}(\boldsymbol{X}')$ and cm' be the corresponding maximum regret and critical machine. If $R_{max}(\boldsymbol{X}')<R_{max}(X)$, set $\boldsymbol{X} = \boldsymbol{X}'$, $R_{max}(\boldsymbol{X}) = R_{max}(\boldsymbol{X}')$, $cm = cm'$, and $N = 1$. If $N = n(n-1)/2$, go to Step 3, otherwise repeat Step 2.

Step 3: The locally optimal robust schedule is $\boldsymbol{X}$ with maximum regret $R_{max}(\boldsymbol{X})$.

4.2 *Simulated annealing algorithm*

Simulated annealing (SA) attempts to improve LS by allowing occasional acceptance of non-improving neighbours with some probability, which decreases as the process evolves (Bertsimas and Tsitsiklis 1997; Chang, Damodaran, and Melouk 2004). We apply the acceptance probability function introduced by Kirkpatrick, Gelatt, and Vecchi (1983) $P(accept) = e^{-\Delta/T}$, where $T = \beta/h$ is the temperature that varies with iteration index h ($\beta = 300$ in this study), and Δ is the increase in maximal regret. If $P(accept)>r$, the neighbour schedule with the worse maximum regret is accepted, where r is a random number between 0 and 1. T controls the possibility of the acceptance of a worse neighbour schedule: when T is large (in early iterations), most of the interchange schedules are accepted; when T becomes small (in later iterations), almost all deteriorating neighbour schedules are rejected. The SA algorithm terminates if there is no improvement after $n(n-1)/2$ iterations.

Procedure of SA

Step 0: Let optimal schedule $\boldsymbol{X}$ under the mid-point scenario be the initial schedule. Determine the maximal regret $R_{max}(\boldsymbol{X})$ and the critical machine cm for schedule $\boldsymbol{X}$ under its worst-case scenario on the basis of Property 2. If $R_{max}(\boldsymbol{X}) = 0$, go to step 3. Let $\boldsymbol{X^B}$ denote the best schedule we have obtained thus far and $R_{max}(\boldsymbol{X}^B)$ be its maximum regret. Set $\boldsymbol{X}^B = \boldsymbol{X}$, $R_{max}(\boldsymbol{X}^B) = R_{max}(\boldsymbol{X})$. Set $k = 0, N = 0$.

Step 1: Set $k = k + 1$. If $k = n$, then set $k = 1$.

Step 2: Set $j = k + 1, N = N + 1$. If $j>n$, go to Step 1. If jobs j and k are on the same machine or neither of them is on the critical machine cm, repeat Step 2. Otherwise, construct a new schedule $\boldsymbol{X}'$ is constructed by interchanging jobs j and k in $\boldsymbol{X}$. Determine the maximum regret $R_{max}(\boldsymbol{X}')$ and critical machine cm' for $\boldsymbol{X}'$ under its worst-case scenario. If $R_{max}(\boldsymbol{X}')<R_{max}(\boldsymbol{X})$, set $\boldsymbol{X} = \boldsymbol{X}'$, $R_{max}(\boldsymbol{X}) = R_{max}(\boldsymbol{X}')$, $cm = cm'$, and $N = 1$; otherwise compute $P(accept) = e^{-\Delta/T}$ and generate $r \sim U(0,1)$. If $P(accept)>r$, we set $\boldsymbol{X} = \boldsymbol{X}'$, $R_{max}(\boldsymbol{X}) = R_{max}(\boldsymbol{X}')$, and $cm = cm'$. If $R_{max}(\boldsymbol{X}^B)>R_{max}(\boldsymbol{X})$, set $\boldsymbol{X}^B = \boldsymbol{X}$ and $R_{max}(\boldsymbol{X}^B) = R_{max}(\boldsymbol{X})$. If $N = n(n-1)/2$, go to Step 3; otherwise, repeat Step 2.

Step 3: The schedule generated by this approach is $\boldsymbol{X}^B$ with maximal regret $R_{max}(\boldsymbol{X}^B)$.

LS and SA should determine the maximal regret at each iteration, which is time-consuming because of the NP-hardness of $P||C_{max}$. If we replace the optimal solution to $P||C_{max}$ with a near-optimal solution obtained by heuristic methods, the computational cost may be significantly reduced. By now, several polynomial-time heuristics have been proposed to tackle the deterministic $P||C_{max}$ problem; these include the longest processing time procedure (LPT) (Graham 1969), pairwise interchange algorithm (PI) (Ghomi and Ghazvini 1998), genetic algorithm (GA) (Min and Cheng 1999), LPT and multifit procedure (LISTFIT) (Gupta and Ruiz-Torres 2001), and SA (Lee, Wu, and Chen 2006). The quality of near-optimal solutions directly affects the accuracy of the maximal regret used in our algorithm. Therefore, SA and PI algorithms, which are deemed more accurate than others (Lee, Wu, and Chen 2006), are used to solve $P||C_{max}$. Four heuristics with near-optimal solutions to $P||C_{max}$ are developed: PI-LS, PI-SA, SA-LS, and SA-SA.

5. Computational experimentation

To assess the performance of the proposed algorithms (summarised in Table 1), extensive computational experiments are carried out. All procedures are coded in MATLAB R2010 and implemented on an Intel Pentium IV 3.00 GHz/512 MB machine.

The first set of experiments involves test problems with $m = 3, 4, 5$ machines and $n = 9, 12, 15$ jobs, which are selected from Mokotoff (2004). We follow the approach in Daniels and Kouvelis (1995) to generate the intervals of job-processing times. For each job, the lower and upper bounds of the processing time are chosen at random from two uniform distributions of integers on intervals $\underline{p}_i \in [10, 50\,\beta_1]$ and $\bar{p}_i \in [\underline{p}_i, \underline{p}_i(1+\beta_2)]$. β_1 and β_2 are two parameters that control the variability of processing time across and within jobs in a given test problem. Five values $(0.2, 0.4, 0.6, 0.8, 1.0)$ are used for both β_1 and β_2. For each combination of m, n, β_1, and β_1, 20 time intervals of each job are generated, resulting in a total of 4500 test problems. Each test problem is solved using the proposed algorithms in Table 1. To illustrate the efficiency of the two improved exact algorithms, we also present the experimental results of the generic iterative relaxation procedure described in Section 3. The optimal solutions to the deterministic $P||C_{max}$ problem or RPMS-relaxed problem are both obtained using the mixed-integer linear programming solver of ILOG CPLEX 12.0. The maximum CPU time allowed is set to 3600 s for exact algorithms and 1200 s for heuristics.

Table 2 shows the computational performance of the proposed algorithms, i.e. the mean CPU time (in seconds) of every algorithm and the average number of iterations required to obtain the robust optimal solutions. The results are summarised over different values of β_2 and aggregated by n to observe the effects of processing time variability and

Table 1. List of algorithms used in this study.

Algorithms		Algorithms for RPMS	Algorithms for $P\|\|C_{max}$
Exact algorithms	IR-U	IR-U	Mixed-integer programming (optimal solution)
	IR-S	IR-S	
Heuristics	LS	LS	
	SA	SA	
	PI-LS	LS	PI
	PI-SA	SA	
	SA-LS	LS	SA
	SA-SA	SA	

Note: IR (general iterative relaxation algorithm) is used for comparison with IR-U and IR-S.

Table 2. Computational performance of solution procedures.

		IR		IR-U[1330]		IR-S		LS	SA[420]	PI-LS	PI-SA	SA-LS	SA-SA
n	β_2	Iteration	CPU	Iteration	CPU	Iteration	CPU	CPU	CPU	CPU	CPU	CPU	CPU
9	0.2	12.0	25.05	10.9	22.33	27.4	17.43	29.44	264.18[1]	2.02	3.58	5.14	43.23
	0.4	13.4	28.34	12.2	25.46	32.3	21.44	29.93	270.83[12]	2.03	3.45	4.96	38.44
	0.6	15.2	32.53	14.0	29.60	36.0	24.73	30.85	260.76[10]	2.04	3.35	4.88	33.67
	0.8	16.7	36.06	15.5	33.14	39.0	27.36	30.65	245.91[15]	2.04	3.25	4.68	30.01
	1.0	18.6	40.67	17.3	37.64	43.5	30.97	31.73	247.02[11]	2.02	3.19	4.55	26.50
Average		15.2	32.53	14.0	29.64	35.6	24.38	30.52	257.74	2.03	3.36	4.84	34.37
12	0.2	54.2	199.73	49.7	172.09	92.8	156.84	57.55	408.72[12]	2.23	4.88	8.93	81.82
	0.4	52.6	210.76	47.3	175.88	102.9	213.30	58.84	487.59[17]	2.25	5.06	8.91	84.01
	0.6	57.5	233.67	52.0	197.89	115.1	271.48	60.93	480.26[12]	2.28	4.89	8.85	75.12
	0.8	64.8	283.77	59.4	247.08	129.9	328.30	63.28	451.90[12]	2.28	4.85	8.85	67.73
	1.0	69.4	318.45	64.0	278.63	138.5	368.77	65.20	413.31[2]	2.29	4.46	8.97	59.62
Average		59.7	249.28	54.5	214.32	115.8	267.74	61.16	448.36	2.27	4.83	8.90	73.66
15	0.2	_	_	47.2	1297.90[181]	_	_	97.84	639.06[73]	2.51	6.86	13.57	122.24
	0.4	_	_	50.4	2226.98[261]	_	_	102.21	688.28[77]	2.47	7.09	13.31	127.87
	0.6	_	_	48.4	2522.69[288]	_	_	103.43	671.74[64]	2.48	6.98	13.66	121.23
	0.8	_	_	_	_[300]	_	_	105.08	690.44[60]	2.51	6.58	13.63	107.66
	1.0	_	_	_	_[300]	_	_	106.53	641.42[42]	2.54	6.26	13.76	99.57
Average		_	_	48.7	2015.85	_	_	103.02	666.19	2.50	6.75	13.59	115.71

Note: The superscript denotes the number of problems unsolved by the corresponding algorithm.

problem size on computational performance. Table 2 shows that the CPU time of each exact algorithm increases rapidly with the number of jobs (n) and rises modestly with increasing values of β_2. IR-U solves all the problems with less running time compared with IR. Moreover, IR-S solves small-size problems with nine jobs more efficiently than IR-U, but performs worse than IR on the problems with 12 jobs. This indicates that IR-U performs better than IR and IR-S when $n \geq 12$, and thus IR-U is used to derive the exact solutions of the problems with 15 jobs. In our computational experimentation, only 170 out of 1500 problems are solved using IR-U in 1 h for $n = 15$, and the 1330 unsolved problems include all the problems with $\beta_2 = 0.8$ and $\beta_2 = 1.0$. All the heuristics can be divided into three levels (L1: SA; L2: LS, and SA-SA; L3: PI-LS, SA-LS, and PI-SA) according to their CPU times. SA performs no better than the exact algorithms until n is up to 15, and a significant number of problems cannot be solved using this algorithm even when $n = 9$. F1 shows how the CPU time of the heuristics in L2 and L3 changes as the problem size (β_2, m, n) increases. In L2, LS performs better than SA-SA. In L3, PI-LS consumes less CPU time, whereas SA-LS requires more CPU time to reach the local optimum. As the number of jobs and number of machines increase, the CPU time of each heuristic in L2 or L3 grows modestly.

To measure the quality of the heuristic solutions, the average and maximum percentage increases above the optimum for each heuristic are presented in Table 3. The results indicate that the heuristic solutions all closely approximate the optimal solutions obtained by the exact algorithms. According to this result, the heuristics can be divided into three grades: G1 (SA), G2 (LS, SA-LS, SA-SA), and G3 (PI-LS, PI-SA). The solution quality of the optimal schedule under

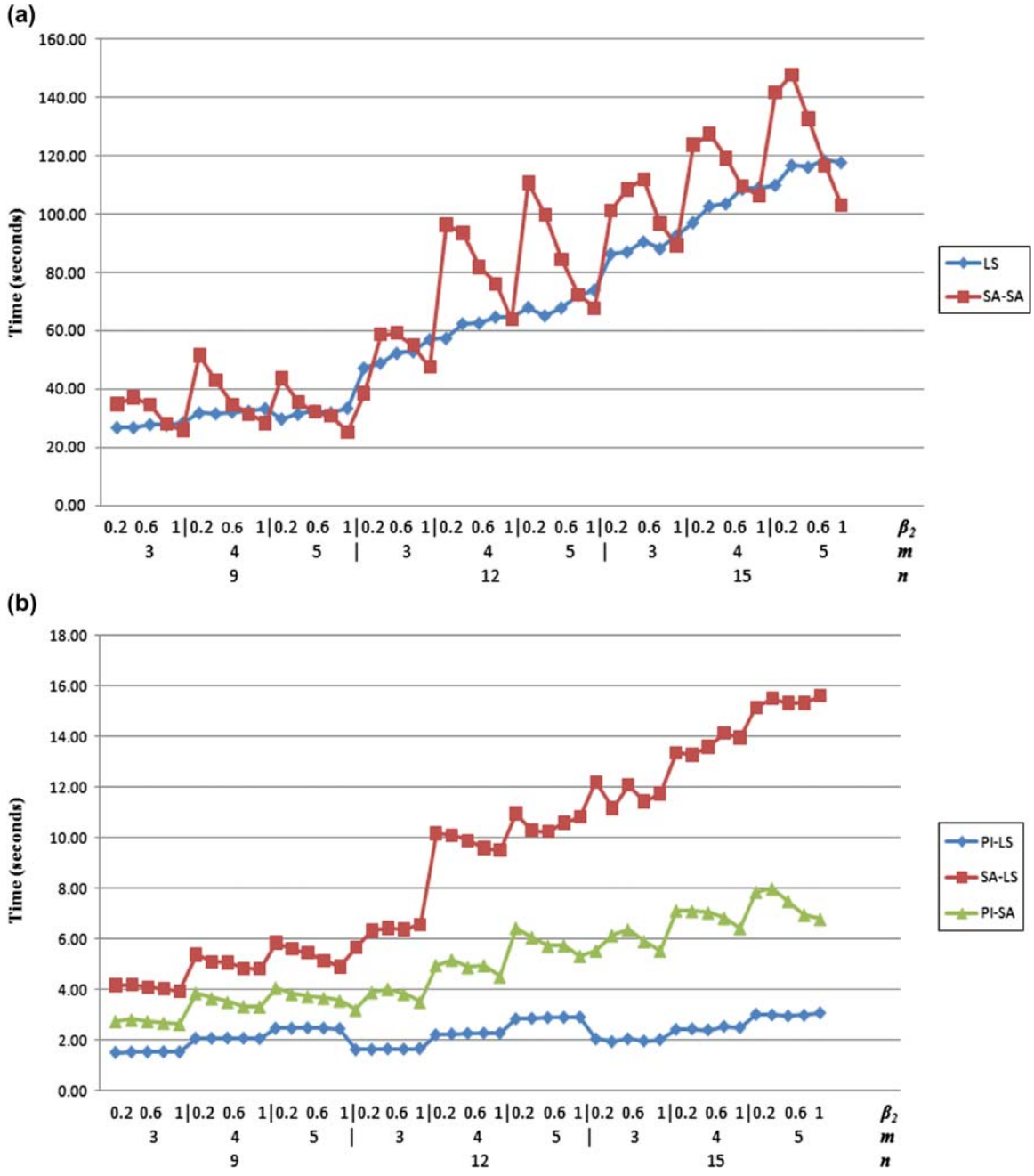

Figure 1. CPU time of the heuristics in L2 (a) and L3 (b).

the mid-point scenario ('mid-op' in Table 3) is lower, with the optimal solutions generated in 53.60% of the test problems and an average percentage deviation of 9.79. The relatively poor performance emphasises the importance of considering processing time variability rather than focusing only on point estimates of processing times in constructing a robust schedule. The last two columns of Table 3 provide information on the expected makespan of the robust optimal schedule compared with the optimal expected makespan under the mid-point scenario. On the 3170 test problems that can be solved using exact algorithms within the time limit, the average percentage deviation of the expected makespan of the robust optimal schedule is only 1.29. A low average percentage deviation indicates that robust optimal schedules hedge against processing time uncertainty while maintaining excellent expected makespan performance.

Table 3. Heuristic solution quality as a percentage above the optimal solution.

		LS		SA		PI-LS		PI-SA		SA-LS		SA-SA		Mid-op		Expected	
n	β_2	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
9	0.2	4.5	100.0	1.0	150.0	6.1	200.0	7.9	200.0	2.2	166.7	2.0	166.7	18.8	800.0	0.4	4.5
	0.4	1.9	66.7	0.6	50.0	2.9	71.4	3.1	71.4	1.3	71.4	1.1	71.4	8.9	200.0	0.9	7.0
	0.6	1.5	40.0	0.6	27.3	2.9	50.0	2.9	40.0	1.7	36.4	1.6	57.1	6.9	240.0	1.3	11.7
	0.8	2.1	112.5	0.8	75.0	2.8	40.0	3.3	50.0	1.7	40.0	2.0	50.0	6.6	155.6	1.7	14.3
	1.0	1.4	40.0	0.6	33.3	2.3	53.8	2.1	53.8	1.8	53.8	1.6	53.8	6.8	166.7	2.3	24.3
Mean [max]		2.3	112.5	0.7	150.0	3.4	200.0	3.8	200.0	1.7	166.7	1.7	166.7	9.6	800.0	1.3	24.3
12	0.2	2.9	40.0	2.2	125.0	10.3	116.7	10.3	100.0	4.2	100.0	3.6	100.0	8.7	125.0	0.6	5.5
	0.4	1.5	22.2	1.6	40.0	7.1	83.3	6.6	75.0	3.4	66.7	3.4	66.7	7.1	100.0	1.0	7.1
	0.6	1.4	16.7	1.6	25.0	5.8	42.1	5.3	45.5	3.2	33.3	2.8	33.3	6.5	33.3	1.3	9.9
	0.8	1.4	20.0	1.4	20.0	5.1	57.1	4.8	43.8	3.1	30.0	2.4	31.3	6.5	40.0	1.3	7.1
	1.0	1.0	12.0	1.0	13.3	4.7	33.3	5.4	37.5	3.2	27.8	4.1	37.5	6.3	50.0	1.5	8.0
Mean [max]		1.6	40.0	1.6	125.0	6.6	116.7	6.5	100.0	3.4	100.0	3.3	100.0	7.0	125.0	1.1	9.9
15	0.2	7.0	40.0	5.8	40.0	13.0	57.1	14.4	57.1	9.5	55.6	8.6	50.0	11.4	50.0	0.8	3.4
	0.4	6.7	25.0	3.6	22.2	9.9	22.2	10.7	30.0	8.5	25.0	5.5	33.3	14.9	44.4	1.6	3.4
	0.6	7.1	14.3	4.9	10.0	14.3	50.0	16.0	50.0	13.1	50.0	15.0	50.0	18.0	30.0	2.4	4.4
	0.8	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	1.0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Mean [max]		7.0	40.0	4.8	40.0	12.4	57.1	13.7	57.1	10.3	55.6	9.7	50.0	14.8	50.0	1.6	4.4

Table 4. Summary of the results for small- and medium-scale problems.

Algorithms	IR-U	LS	SA	PI-LS	PI-SA	SA-LS	SA-SA
T (s)	559.02	64.90	457.43	2.27	4.98	9.11	74.58
PD (%)	0	3.12	1.97	6.72	7.14	4.37	4.13
OP (%)	100	81.77	87.34	65.24	65.33	77.07	79.05

Note: T, CPU time; PD, percentage deviation from optimum; OP, percentage of optimal solutions.

Table 5. CPU time rank and solution quality rank of the proposed heuristics.

	Heuristics Ranks
T	PI-LS≻PI-SA≻SA-LS≻LS≻SA-SA ≻SA
OP(PD)	SA≻LS≻SA-SA≻SA-LS≻PI-SA≻PI-LS

Note: T, CPU time; PD, percentage deviation from optimum; OP, percentage of optimal solutions.

Table 6. Computational results of the selected heuristics for large problems ($\beta_1, \beta_2 = 0.5$).

		LS		SA-LS		PI-LS	
n	m	T (s)	PD (%)	T (s)	PD (%)	T (s)	PD (%)
20	3	133.17	0.92	16.17	0.90	2.07	0.91
	4	176.71	1.61	30.96	1.12	2.75	2.18
	5	194.77	1.02	30.81	1.85	3.79	3.46
Mean (∼20)		168.22	1.18	25.98	1.29	2.87	2.18
25	3	*	*	30.18	0.00	2.59	0.00
	4	*	*	40.67	1.31	3.42	0.23
	5	*	*	50.83	0.53	4.31	3.71
30	3	*	*	41.54	0.58	3.28	0.17
	4	*	*	66.60	0.75	4.49	1.39
	5	*	*	70.24	0.59	5.30	1.84
40	3	*	*	92.98	0.35	5.66	0.10
	4	*	*	114.99	0.74	6.53	0.65
	5	*	*	158.55	0.49	7.56	1.03
50	3	*	*	171.54	0.44	9.15	0.34
	4	*	*	233.10	0.67	10.52	1.07
	5	*	*	278.81	0.66	11.54	0.66
Mean (25∼50)		*	*	112.50	0.59	6.19	0.93

Note: An asterisk indicates that at least one problem instance is unsolved within 600 s.

The performance of the proposed algorithms for small- and medium-scale problems are summarised in Table 4, which shows the average CPU time, percentage deviations from the optimum, and percentage of optimal solutions identified for the test problems.

On the basis of the CPU time consumed and the solution quality of the proposed heuristics, we present the ranks of heuristics in Table 5, and recommend LS, PI-LS, and SA-LS to solve large-scale problems. In general, SA is inferior because it takes more CPU time. LA performs better than SA-SA in terms of both CPU time and solution quality. PI-LS is chosen instead of PI-SA because it consumes less CPU time while maintaining the same grade of solution quality (G3). SA-LS (in G2 and L3) is recommended because it quickly identifies higher-quality solutions.

To compare the performance of the three selected heuristics when solving large-scale problems, a second set of experiments is designed. We generate the problem instances with $n = 20, 25, 30, 40$, and 50 jobs, and $m = 3, 4, 5$ machines. The CPU times of LS, SA-LS, and PI-LS change little with increasing β_1 and β_2 values; thus, we choose only $\beta_1, \beta_2 = 0.5$ to generate the interval processing times of jobs. For each combination of m, n, β_1 and β_2, 20 time

intervals of each job are generated, resulting in 300 test problems. The results are summarised in Table 6, which shows the mean CPU times (T) and average percentage deviations (PD) from the best solutions of the three selected heuristics. LS solves problems with up to 20 jobs. PI-LS takes far less time than SA-LS in identifying a solution, but its solution quality is slightly inferior to that of SA-LS. We reach the same conclusion as that drawn in the experiments of small- and medium-scale problems.

6. Conclusions

Parallel machine scheduling is commonly encountered in the manufacturing environment and has been extensively studied. In the most of the literature, the job-processing times are assumed to be fixed and precisely known. In practice, however, the processing times of jobs are unknown in advance and their exact probability distributions are often difficult to evaluate because of limited information especially for the set of jobs that are processed for the first time. This paper discusses a min–max regret (robust) version of $P||C_{max}$ scheduling problem (RPMS) with interval job-processing times. To solve this NP-hard problem, we first propose Properties 1 and 2, which show that the regret-maximising scenario exists only among a finite number of extreme point scenarios. Two exact algorithms (IR-U and IR-S) are then developed on the basis of a general IR algorithm to obtain the robust optimal solution of the RP MS problem. Moreover, a good initial schedule (optimal schedule under a mid-point scenario) is discussed, and an upper bound on its maximal regret is presented in Property 3. Decision-makers can directly use the optimal schedule under the mid-point scenario and avoid the time-consuming procedure of identifying a robust optimal schedule if this upper bound is acceptable. Additionally, efficient heuristics, such as LS and SA, are developed to solve large-scale problems.

From the computational results for small- and medium-scale problems, we draw the following conclusions: (1) IR-S solves small-size problems more efficiently than IR-U, but performs worse than IR-U on medium-size problems. (2) The heuristic algorithms can be divided into three levels according to their CPU times (L3 $\succ$ L2 $\succ$ L1, L1: SA; L2: LS, SA-SA; L3: PI-LS, SA-LS, and PI-SA), and into three grades according to their solution qualities (G1$\succ$G2 $\succ$ G3, G1: SA; G2: LS, SA-SA, SA-LS; G3: PI-SA, PI-LS). (3) The robust optimal schedules hedge effectively against processing time uncertainty while maintaining excellent expected makespan performance. For large problems, SA-LS, PI-LS, and LS are recommended after comparing the CPU time and solution quality of these heuristics. Our computational results for large-scale problems demonstrate that LS can solve instances with only up to 20 jobs within a given time limit. Both SA-LS and PI-LS can identify heuristic solutions for all problem instances within the time limit, and SA-LS outperforms PI-LS in terms of solution quality but at the expense of longer running times.

The proposed exact algorithms are computationally expensive because RPMS-relaxed is a mixed-integer linear programming problem, whose difficulty increases with the number of regret cuts added into it. Future research can improve exact algorithms by dropping inactive regret cuts at each iteration or creating new types of regret cuts [note: we register failure when using cut 2 in Aissi, Bazgan, and Vanderpooten (2009)]. Moreover, more efficient exact algorithms that can solve the deterministic $P||C_{max}$ problem may increase the size of problems that can be solved by our algorithms. The framework presented in this paper for the robust version of $P||C_{max}$ scheduling problem serves as an effective approach to handling uncertainties in actual manufacturing environments and establishes the academic groundwork for other parallel machine-scheduling problems with interval data.

Acknowledgement

The authors are grateful for financial support from the National Natural Science Foundation of China (Grants 71072128 and 71001084).

References

Aissi, H., C. Bazgan, and D. Vanderpooten. 2005a. “Approximation Complexity of Min–Max (Regret) Versions of Shortest Path, Spanning Tree, and Knapsack.” *Algorithms - Esa* 2005 (3669): 862–873.

Aissi, H., Bazgan, C. and Vanderpooten, D. 2005b. “Complexity of the Min–Max and Min–Max Regret Assignment Problems.” *Operations Research Letters* 33(6), 634–640.

Aissi, H., C. Bazgan, and D. Vanderpooten. 2009. “Min–Max and Min–Max Regret Versions of Combinatorial Optimization Problems: A Survey.” *European Journal of Operational Research* 197 (2): 427–438.

Allahverdi, A., and H. Aydilek. 2010. “Heuristics for the Two-Machine Flowshop Scheduling Problem to Minimise Makespan with Bounded Processing times.” *International Journal of Production Research* 48 (21): 6367–6385.

Anglani, A., A. Grieco, E. Guerriero, and R. Musmanno. 2005. “Robust Scheduling of Parallel. Machines with Sequence-Dependent Set-up Costs.” *European Journal of Operational Research* 161 (3): 704–720.

Averbakh, I. 2006. "The Minmax Regret Permutation Flow-Shop Problem with Two Jobs." *European Journal of Operational Research* 169 (3): 761–766.

Bertsimas, D., and J.N. Tsitsiklis. 1997. *Introduction to Linear Optimization*. USA: Athena Scientific.

Chang, P. T., K. P. Lin, P. F. Pai, C. Z. Zhong, C. H. Lin, and L. T. Hung. 2008. "Ant Colony Optimization System for a Multi-Quantitative and Qualitative Objective Job-Shop Parallel-Machine-Scheduling Problem." *International Journal of Production Research* 46 (20): 5719–5759.

Chang, P. Y., P. Damodaran, and S. Melouk. 2004. "Minimizing Makespan on Parallel Batch Processing Machines." *International Journal of Production Research* 42 (19): 4211–4220.

Chen, C. L. 2008. "Bottleneck-Based Heuristics to Minimize Tardy Jobs in a Flexible Flow Line with Unrelated Parallel Machines." *International Journal of Production Research* 46 (22): 6415–6430.

Chien, C. F., and C. H. Chen. 2007. "Using Genetic Algorithms (Ga) and a Coloured Timed Petri Net (Ctpn) for Modelling the Optimization-Based Schedule Generator of a Generic Production Scheduling System." *International Journal of Production Research* 45 (8): 1763–1789.

Chuang, M. C., C. J. Liao, and C. W. Chao. 2010. "Parallel Machine Scheduling with Preference of Machines." *International Journal of Production Research* 48 (14): 4139–4152.

Chung, S. H., Y. T. Tai, and W. L. Pearn. 2009. "Minimising Makespan on Parallel Batch Processing Machines with Non-Identical Ready Time and Arbitrary Job Sizes." *International Journal of Production Research* 47 (18): 5109–5128.

Conde, E. 2009. "A Minmax Regret Approach to the Critical Path Method with Task Interval Times." *European Journal of Operational Research* 197 (1): 235–242.

Daniels, R. L., and P. Kouvelis. 1995. "Robust Scheduling to Hedge Against Processing Time Uncertainty in Single-Stage Production." *Management Science* 41 (2): 363–376.

Deineko, V. G., and G. J. Woeginger. 2010. "Pinpointing the Complexity of the Interval Min–Max Regret Knapsack Problem." *Discrete Optimization* 7 (4): 191–196.

Gamberini, R., F. Lolli, B. Rimini, M. Torelli, and E. Castagnetti. 2011. "An Innovative Approach for Job Pre-Allocation to Parallel Unrelated Machines in the Case of a Batch Sequence Dependent Manufacturing Environment." *International Journal of Production Research* 49 (14): 4353–4376.

Ghomi, S. M. T. F., and F. J. Ghazvini. 1998. "A Pairwise Interchange Algorithm for Parallel Machine Scheduling." *Production Planning & Control* 9 (7): 685–689.

Graham, R. L. 1969. "Bounds on Multiprocessing Timing Anomalies." *SIAM Journal on Applied Mathematics* 17 (2): 416–429.

Gupta, J. N. D., and J. Ruiz-Torres. 2001. "A Listfit Heuristic for Minimizing Makespan on Identical Parallel Machines." *Production Planning & Control* 12 (1): 28–36.

Hu, X. F., J. S. Bao, and Y. Jin. 2010. "Minimising Makespan on Parallel Machines with Precedence Constraints and Machine Eligibility Restrictions." *International Journal of Production Research* 48 (6): 1639–1651.

Inuiguchi, M., and M. Sakawa. 1995. "Minimax Regret Solution to Linear-Programming Problems with an Interval Objective Function." *European Journal of Operational Research* 86 (3): 526–536.

Józefowska, J., M. Mika, R. Rozycki, G. Waligora, and J. Weglarz. 1998. "Local Search Metaheuristics for Discrete-Continuous Scheduling Problems." *European Journal of Operational Research* 107 (2): 354–370.

Jia, J., and S. J. Mason. 2009. "Semiconductor Manufacturing Scheduling of Jobs Containing Multiple Orders on Identical Parallel Machines." *International Journal of Production Research* 47 (10): 2565–2585.

Kaczmarczyk, W. 2011. "Proportional Lot-Sizing and Scheduling Problem with Identical Parallel Machines." *International Journal of Production Research* 49 (9): 2605–2623.

Kasperski, A. 2005. "Minimizing Maximal Regret in the Single Machine Sequencing Problem with Maximum Lateness Criterion." *Operations Research Letters* 33 (4): 431–436.

Kasperski, A., and P. Zielinski. 2006. "An Approximation Algorithm for Interval Data Minmax Regret Combinatorial Optimization Problems." *Information Processing Letters* 97 (5): 177–180.

Kasperski, A., and P. Zielinski. 2008. "A 2-Approximation Algorithm for Interval Data Minmax Regret Sequencing Problems with the Total Flow Time Criterion." *Operations Research Letters* 36 (3): 343–344.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. "Optimization by Simulated Annealing." *Science* 220 (4598): 671–680.

Kouvelis, P., R. L. Daniels, and G. Vairaktarakis. 2000. "Robust Scheduling of a Two-Machine Flow Shop with Uncertain Processing Times." *IIE Transactions* 32 (5): 421–432.

Kouvelis, P., and G. Yu. 1997. *Robust Discrete Optimization and its Applications*. Boston: Kluwer Academic Publishers.

Lebedev, V., and I. Averbakh. 2006. "Complexity of Minimizing the Total Flow Time with Interval Data and Minmax Regret Criterion." *Discrete Applied Mathematics* 154 (15): 2167–2177.

Lee, W. C., C. C. Wu, and P. Chen. 2006. "A Simulated Annealing Approach to Makespan Minimization on Identical Parallel Machines." *International Journal of Advanced Manufacturing Technology* 31 (3–4): 328–334.

Lin, J., and T. S. Ng. 2011. "Robust Multi-Market Newsvendor Models with Interval Demand Data." *European Journal of Operational Research* 212 (2): 361–373.

Lu, C. C., Lin, S. W. and Ying, K. C. 2011. Robust Scheduling on a Single Machine to Minimize Total Flow Time. *Computers & Operations Research*, doi:10.1016/j.cor.2011.10.003.

Mason, S. J., S. Jin, and J. Jampani. 2009. "A Moving Block Heuristic to Minimise Earliness and Tardiness Costs on Parallel Machines." *International Journal of Production Research* 47 (19): 5377–5390.

Mausser, H. E., and M. Laguna. 1998. "A New Mixed Integer Formulation for the Maximum Regret Problem." *International Transactions in Operational Research* 5 (5): 389–403.

Mausser, H. E., and M. Laguna. 1999. "A Heuristic to Minimax Absolute Regret for Linear Programs with Interval Objective Function Coefficients." *European Journal of Operational Research* 117 (1): 157–174.

Min, L., and W. Cheng. 1999. "A Genetic Algorithm for Minimizing the Makespan in the Case of Scheduling Identical Parallel Machines." *Artificial Intelligence in Engineering* 13 (4): 399–403.

Mokotoff, E. 2004. "An Exact Algorithm for the Identical Parallel Machine Scheduling Problem." *European Journal of Operational Research* 152 (3): 758–769.

Montemanni, R. 2007. "A Mixed Integer Programming Formulation for the Total Flow Time Single Machine Robust Scheduling Problem with Interval Data." *Journal of Mathematical Modelling and Algorithms* 6 (2): 287–296.

Montemanni, R., and L. Gambardella. 2005. "The Robust Shortest Path Problem with Interval Data Via Benders Decomposition. *4OR: A Quarterly Journal of*." *Operations Research* 3 (4): 315–328.

Montemanni, R., and L. M. Gambardella. 2004. "An Exact Algorithm for the Robust Shortest Path Problem with Interval Data." *Computers & Operations Research* 31 (10): 1667–1680.

Montemanni, R., and L. M. Gambardella. 2005. "A Branch and Bound Algorithm for the Robust Spanning Tree Problem with Interval Data." *European Journal of Operational Research* 161 (3): 771–779.

Monternanni, R. 2006. "A Benders Decomposition Approach for the Robust Spanning Tree Problem with Interval Data." *European Journal of Operational Research* 174 (3): 1479–1490.

Ozlen, M., and M. Azizoglu. 2009. "Generating all Efficient Solutions of a Rescheduling Problem on Unrelated Parallel Machines." *International Journal of Production Research* 47 (19): 5245–5270.

Ozlen, M., and S. Webster. 2010. "Minimising Total Flow-Time on Two Parallel Machines with Planned Downtimes and Resumable Jobs." *International Journal of Production Research* 48 (1): 201–226.

Shin, H. J., and Y. H. Kang. 2010. "A Rework-Based Dispatching Algorithm for Module Process in Tft-Lcd Manufacture." *International Journal of Production Research* 48 (3): 915–931.

Vanderheyden, L. 1981. "Scheduling Jobs with Exponential Processing and Arrival Times on Identical Proces so as to Minimize the Expected Makespan." *Mathematics of Operations Research* 6 (2): 305–312.

Weber, R. R. 1982. "Scheduling Jobs with Stochastic Processing Requirements on Parallel Machines to Minimize Makespan or Flowtime." *Journal of Applied Probability* 19 (1): 167–182.

Yang, J., and G. Yu. 2002. "On the Robust Single Machine Scheduling Problem." *Journal of Combinatorial Optimization* 6 (1): 17–33.

Appendix 1

Proof of Property 1.

We denote the set containing all the jobs on machine j in schedule $\boldsymbol{X}$ as $P(\boldsymbol{X}, j)$, $j \in M$, i.e. if $x_{ij} = 1$, then $i \in P(\boldsymbol{X}, j)$.

Worst-case scenario $\boldsymbol{s}^0$ can be transformed into scenario $\boldsymbol{s}^f$ by decreasing $p_i^{s^0}$ to $\underline{p}_i$ for all $i \notin P(\boldsymbol{X},f)$ and by increasing $p_i^{s^0}$ to $\bar{p}_i$ for all $i \in P(\boldsymbol{X},f)$ f is also the critical machine in $\boldsymbol{X}$ under $\boldsymbol{s}^f$. Let $\Delta = F(\boldsymbol{X}, \boldsymbol{s}^f) - F(\boldsymbol{X}, \boldsymbol{s}^0)$ denote the increase in the makespan under $\boldsymbol{s}^f$. Let $\boldsymbol{X}^*_{s^0}$ be the optimal schedule for scenario $\boldsymbol{s}^0$. Because the makespan in $\boldsymbol{X}^*_{s^0}$ cannot increase by more than Δ under scenario $\boldsymbol{s}^f$ in comparison with $\boldsymbol{s}^0$, we get:

$$F^*_{\boldsymbol{s}^f} - F^*_{s^0} \leq F\left(\boldsymbol{X}^*_{s^0}, \boldsymbol{s}^f\right) - F\left(\boldsymbol{X}^*_{s^0}, \boldsymbol{s}^0\right) \leq \Delta = F\left(\boldsymbol{X}, \boldsymbol{s}^f\right) - F\left(\boldsymbol{X}, \boldsymbol{s}^0\right).$$

From the above-mentioned inequality, we have $F(\boldsymbol{X}, \boldsymbol{s}^0) - F^*_{s^o} \leq F(\boldsymbol{X}, \boldsymbol{s}^f) - F^*_{\boldsymbol{s}^f}$, indicating that the maximal regret cannot decrease if we replace $\boldsymbol{s}^0$ with $\boldsymbol{s}^f$, and which is also the worst-case scenario for $\boldsymbol{X}$.

Proof of Property 2.

Equation (11) shows that

$$R_{max}(\boldsymbol{X}) \leq \max_{j \in M} \left\{ \sum_{i=1}^{n} \bar{p}_i x_{ij} - F^*_{s^j} \right\} \tag{P2-1}$$

Suppose by contraction that there exists a machine $k \in M$ such that

$$R_{max}(\boldsymbol{X}) < \sum_{i=1}^{n} \bar{p}_i x_{ik} - F^*_{\boldsymbol{s}^k}$$

Since $F(\boldsymbol{X}, \boldsymbol{s}^k) \geq \sum_{i=1}^{n} \bar{p}_i x_{ik}$, we get

$$R_{max}(\boldsymbol{X}) < F(\boldsymbol{X}, \boldsymbol{s}^k) - F^*_{\boldsymbol{s}^k}$$

which contradicts the definition of $R_{max}(\boldsymbol{X})$. Therefore, the equality in (P2-1) holds, and the proof is complete.

Proof of Property 3.

Because $\bar{p}_i = (1+\alpha_i)\underline{p}_i$ and $p_i^{s^{\frac{1}{2}}} = (\bar{p}_i + \underline{p}_i)/2$, we can represent $\underline{p}_i$ and $\bar{p}_i$with $p_i^{s^{\frac{1}{2}}}$:

$$\underline{p}_i = \frac{2}{2+\alpha_i} p_i^{s^{\frac{1}{2}}}, i \in J$$

$$\bar{p}_i = \left(1 + \frac{\alpha_i}{2+\alpha_i}\right) p_i^{s^{\frac{1}{2}}}, i \in J.$$

To simplify the proof, we first define some scenarios:

$\bar{\boldsymbol{s}}$: upper-bound scenario, i.e. $p_i^{\bar{s}} = \bar{p}_i$, for all $i \in J$

$\underline{\boldsymbol{s}}$: lower-bound scenario, i.e. $p_i^{\underline{s}} = \overline{p}_i$, for all $i \in J$

$\bar{s}^\alpha$: α -upper-bound scenario, i.e., $p_i^{\bar{s}^\alpha} = \bar{p}_i^\alpha$, for all $i \in J$, where $\bar{p}_i^\alpha = (1 + \frac{\alpha}{2+\alpha}) p_i^{s^{\frac{1}{2}}} \geq p_i^{\bar{s}}$

$\underline{\boldsymbol{s}}^\alpha$:$\alpha$-lower-bound scenario, i.e. $p_i^{s^\alpha} = \underline{p}_i^\alpha$, for all $i \in J$, where $\underline{p}_i{}^\alpha = \frac{2}{2+\alpha} p_i^{s^{\frac{1}{2}}} \leq p_i^{\underline{s}}$.

Note that the processing times of the jobs under scenario $\underline{\boldsymbol{s}}^\alpha$ or $\boldsymbol{s}^{-\alpha}$ are proportional to the processing times under $\boldsymbol{s}^{\frac{1}{2}}$, so that the optimal schedule under scenarios $\underline{\boldsymbol{s}}^\alpha$ or $\bar{\boldsymbol{s}}^\alpha$ is the same as $\boldsymbol{s}^{\frac{1}{2}}$; i.e. $F(\boldsymbol{X}, \underline{s}^\alpha) = F^*_{\underline{\boldsymbol{s}}^\alpha}, F(\boldsymbol{X}, \bar{\boldsymbol{s}}^\alpha) = F^*_{\bar{s}^\alpha}$ and their minimal makespans satisfy the following relationships:

$$F(\boldsymbol{X}, \bar{\boldsymbol{s}}^\alpha) = \left(1 + \tfrac{\alpha}{2+\alpha}\right) F^*_{s^{\frac{1}{2}}},$$
$$F(\boldsymbol{X}, \underline{\boldsymbol{s}}^\alpha) = \left(\tfrac{2}{2+\alpha}\right) F^*_{s^{\frac{1}{2}}}.$$

Moreover, because $\bar{p}_i^\alpha \geq p_i^{\bar{s}}$ and $\underline{p}_i^\alpha \leq p_i^{\underline{s}}$ for all $i \in J$, we have $F(\boldsymbol{X}, \bar{\boldsymbol{s}}^\alpha) \geq F(\boldsymbol{X}, \bar{\boldsymbol{s}})$ and we have $F(\boldsymbol{X}, \underline{\boldsymbol{s}}^\alpha) \leq F(\boldsymbol{X}, \underline{\boldsymbol{s}})$.

We denote the worst-case scenario for schedule $\boldsymbol{X}$ as $\boldsymbol{s}^0$. According to the definition of the maximum regret, we obtain the following inequality:

$$\begin{aligned}
R_{max}(\boldsymbol{X}) &= F(\boldsymbol{X}, \boldsymbol{s}^0) - F^*_{s^0} \\
&\leq F(\boldsymbol{X}, \bar{\boldsymbol{s}}) - F^*_{\underline{\boldsymbol{s}}} \\
&\leq F(\boldsymbol{X}, \bar{\boldsymbol{s}}^\alpha) - F^*_{\underline{s}^\alpha} \\
&= F(\boldsymbol{X}, \bar{\boldsymbol{s}}^\alpha) - F(\boldsymbol{X}, \underline{\boldsymbol{s}}^\alpha) \\
&= \left(1 + \frac{\alpha}{2+\alpha}\right) F^*_{s^{\frac{1}{2}}} - \left(\frac{2}{2+\alpha}\right) F^*_{s^{\frac{1}{2}}} \\
&= \frac{2\alpha}{2+\alpha} F^*_{s^{\frac{1}{2}}}
\end{aligned}$$

Proof of Property 4.

The proof of Property 4 is clear when two machines are considered. Therefore, we need to consider only the case where the number of machines is greater than two.

Let $\boldsymbol{X}$ be a schedule in which job h is on a non-critical machine under its worst-case scenario $\boldsymbol{s}^0$. We construct a new schedule $\boldsymbol{X}'$ by moving job h to another machine, and assume by contradiction that the maximum regret of $\boldsymbol{X}'$ is less than that of $\boldsymbol{X}$. Let $\boldsymbol{s}'$ denote the worst-case scenario for $\boldsymbol{X}'$. We have:

$$R(\boldsymbol{X}, \boldsymbol{s}^0) > R(\boldsymbol{X}', \boldsymbol{s}') \geq R(\boldsymbol{X}', \boldsymbol{s}^0). \tag{P4-1}$$

Given that job h is moved from a non-critical machine, the makespan in schedule $\boldsymbol{X}'$ cannot be lower than that in $\boldsymbol{X}$ under scenario $\boldsymbol{s}^0$; i.e.

$$F(\boldsymbol{X}', \boldsymbol{s}^0) \geq F(\boldsymbol{X}, \boldsymbol{s}^0).$$

From the above-mentioned results, we have:

$$R(\boldsymbol{X}', \boldsymbol{s}^0) = F(\boldsymbol{X}', \boldsymbol{s}^0) - F^*_{\boldsymbol{s}^0} \geq F(\boldsymbol{X}, \boldsymbol{s}^0) - F^*_{\boldsymbol{s}^0} = R(\boldsymbol{X}, \boldsymbol{s}^0)$$

which contradicts (P4-1).

Proof of Property 5.

This proof is clear when two machines are considered. Therefore, we need to consider only the cases where there are more than two machines.

Consider a schedule $\boldsymbol{X}$ and let $\boldsymbol{s}^0$ denote its worst-case scenario. We construct a new schedule $\boldsymbol{X}'$ by interchanging the positions of jobs k and j, which are on two different non-critical machines in $\boldsymbol{X}$ under scenario $\boldsymbol{s}^0$. Suppose by contraction that the maximum regret of $\boldsymbol{X}'$ is less than that of $\boldsymbol{X}$. Let $\boldsymbol{s}'$ denote the worst-case scenario for $\boldsymbol{X}'$. We have:

$$R(\boldsymbol{X}, \boldsymbol{s}^0) > (\boldsymbol{X}', \boldsymbol{s}') \geq (\boldsymbol{X}', \boldsymbol{s}^0) \tag{P5-1}$$

Because the jobs on the critical machine in schedule $\boldsymbol{X}$ under scenario $\boldsymbol{s}^0$ are not changed in schedule $\boldsymbol{X}'$, the makespan in schedule $\boldsymbol{X}'$ cannot be lower than that in $\boldsymbol{X}$; i.e.

$$F(\boldsymbol{X}', \boldsymbol{s}^0) \geq F(\boldsymbol{X}, \boldsymbol{s}^0).$$

From the above-mentioned results, we have:

$$R(\boldsymbol{X}', \boldsymbol{s}^0) = F(\boldsymbol{X}', \boldsymbol{s}^0) - F^*_{\boldsymbol{s}^0} \geq F(\boldsymbol{X}, \boldsymbol{s}^0) - F^*_{\boldsymbol{s}^0} = R(\boldsymbol{X}, \boldsymbol{s}^0)$$

which contradicts (P5-1).