Robustness Analysis for the Re-entrant Flow Shop Problem

Alexandru Bobe
Supervisor(s): Eghonghon Eigbe, Neil Yorke-Smith
EEMCS, Delft University of Technology, The Netherlands

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 19, 2022

Robustness Analysis for the Re-entrant Flow Shop Problem

Alexandru Bobe[1]
Supervisor(s): Eghonghon Eigbe[1] , **Neil Yorke-Smith**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

Abstract

Scheduling is required in almost every industry and when done well it can bring a lot of revenue. Flexibility is often forgotten when creating the initial schedules. Therefore, in case of an unexpected delay, the whole schedule has to suffer. In this paper, we consider a re-entrant flow shop with sequence-dependent setup times and relative due dates for our industrial partner, which specialises in industrial printers. Then, we perform a robustness analysis on real schedules from the industry, which can be extended to any system represented as a flow shop with relative due dates. We find how much time a schedule with relative due dates has before it becomes infeasible. We continue by empirically creating a new robustness measure and comparing it with state-of-the-art techniques. The experiments confirm that this measure can be useful in creating robust initial schedules for re-entrant flow shops with added idle time that has a minimal effect on the total duration of the solution.

1 Introduction

Scheduling has always been relevant to the real world and it is now undergoing intense study in many domains, including Artificial Intelligence [1]. Whether it is about humans or machines, an execution plan is beneficial for achieving optimal results. In a small team, a manager can cope with unexpected situations, when resources are insufficient or time is limited. However, when it comes to complex manufacturing systems, there is a need for sophisticated scheduling strategies. The system must highly optimize the process, by choosing which machines will complete the job and in which specific order, to achieve the best results with the lowest quantity of resources used. In Managerial Sciences, the main characteristic used for comparison is revenue. In Computer Science, we talk about time, as it is tightly connected to revenue and allows for easier generalisation. Modern systems are present in many industries and to name a few, public transport, manufacturing and even aerospace engineering [2].

Much research has been done in collaboration with industrial partners in order to prove the difficulty of scheduling problems [3] and optimize their systems. Our industrial partner is concerned with industrial printing. The printing process is comprised of different operations and various durations, including the printing time, warm-up time, recovery time and loading time. This level of complexity is specific to combinatorial problems, which live at the intersection of Mathematics and Computer Science. A few papers have already explored the optimization of industrial printers using advanced computations [4; 5]. They present the problem as a flow shop since the machine ordering is the same for every job. Moreover, they present some techniques for generating solutions.

The implementation of a schedule in a real-world setting is not a one-fit-all scenario. However, we describe in Figure 1 a scenario that can be applied in various environments. First, an initial schedule has to be created. Various techniques are being used to do it, including Bellman-Ford and genetic algorithms [6]. Then, the initial solution is improved in various ways. The schedule is analysed in terms of different characteristics, depending on the overall goal. Besides the total makespan, the robustness of the schedule is also important. At first, as in our paper, this analysis deals with a deterministic schedule. Then, as the schedule is improved, the duration of operations takes stochastic values to better mimic the real world. After the analysis is done, the manager chooses if they want to add idle time to the schedule to improve the overall performance. We deem it a useful step for improving the robustness of schedules and we create the means for efficiently doing it. Afterwards, the schedule is applied in the production environment and the probability distributions are found for the operations in the given schedule, as the operational times might follow a different distribution depending on the chosen sequence. It is a difficult task to find the probability distributions, which have a great impact on the performance of the solution. With these distributions known, an improved solution can be found. There exist no analytical solution for this problem, however, Baker and Altheimer propose some computational efficient heuristics to solve it [7]. Then, the schedule is analysed again and the implementation cycle starts over. This is a never-ending cycle, as machine breakdowns and deterioration are always present in a production system [8] and the latest schedule might not be the most efficient anymore. Xiong et al. propose a robustness measure for stochastic job shops. This measure can also be applied in our case for the schedules with stochastic values. They propose to measure

the robustness of a stochastic schedule using the Monte Carlo method with the following formula:

$$RM = \frac{\frac{1}{N}\sum_{i=1}^{N} M_i(S) - M_0(S)}{M_0(S)} \quad (1)$$

where N is the number of experiments, M_0 is the expected makespan of the schedule and M_i is the actual makespan obtained in the experiment.

In this paper, we are not interested in how the initial schedules were generated, but in assessing if an existing schedule that assumes deterministic values for the duration can indeed take into account delays without totally breaking down or having a minimum increase in the duration. We use a re-entrant flow shop and we perform a robustness analysis on schedules with deterministic values generated for our industrial partner, depicted as the second step in Figure 1. This analysis is valuable not only for our partner but for any system depicted as a flow shop with relative due dates. Then, we demonstrate how our measure compares to state-of-the-art techniques [9; 10]. The goal of this paper is to create a robustness measure that can be used while creating re-entrant flow shop schedules with inserted idle time to improve flexibility while minimising the delay in completion time.

2 Problem Definition

In this section, we define the re-entrant flow shop with sequence-dependent setup times and relative due dates, introduce the terminology and formalize the notations. In order to facilitate the understanding of the reader, the use case is an industrial printer, but it can be generalised to any large production system. It is important to note the difference between scheduling problems or benchmarks and schedules or solutions.

2.1 The Re-entrant Flow Shop Problem

A flow shop is a complex combinatorial problem [11]. A re-entrant flow shop is a scheduling problem where one or more operations must be performed multiple times by the same machine, see Figure 2 for an example visualization. For example, printing both sides of a paper requires the sheet to go twice through a machine. In literature, a re-entrant flow show is usually represented as a tuple [4; 12] with minor variations. We chose to define the tuple $(M, J, r, O, P, S, SS, D, \phi)$ for a schedule with n jobs and m machines. $M = \{m_1, m_2...m_m\}$ is the set of machines. Each machine can only perform a single operation at any given moment and it must fully complete an operation before being able to start another operation. $J = \{j_1, j_2...j_n\}$ is the set of jobs. For our problem, the flow shop has a fixed job order. Different re-entrant flow shops, where the ordering of the jobs is not fixed, have been proven to be NP-complete [13]. r defines the number of operations that has to be completed for each job. $O_i = \{o_{i,1}, o_{i,2}...o_{i,r}\}$ is the set of operations for job j_i. $P = \{p_{1,1}, ..., p_{n,m}\}$ describes the processing times of all the operations in the scheduling problem and it is defined as $P : O \to \mathbb{N}$. S is defined as an integer and describes the default setup of the machines or how much time it takes a machine to prepare for an operation. $SS_{(o_x,o_y)}$ is the sequence-dependent setup times and describes how much time it takes for a machine to prepare for executing operation o_i after previously executing operation o_{i-1}, defined as $SS : O \times O \to N$. In case the SS is not defined for a pair of operations, then the default setup time applies. The sequence-dependent setup time can be exemplified by a printer machine that takes a long time to print a colour page after just previously printing a black and white page. Moreover, these sequence-dependent setup times add to the overall complexity of the problem. D is the set of all the defined relative due dates. $\phi = [m_1, .., m_r]$ is the re-entrance vector which shows on which machine the operation must be performed, being the same for every job and containing r elements.

2.2 Schedule Feasibility and Optimality

The solution to a flow shop is a schedule. This schedule defines all the operations' start times. In order for this schedule to be feasible it has to adhere to some constraints:

- Every machine can only execute a single operation at a given time and a started operation must be completed
- Every machine's setup time, default or sequence-dependent, must be allocated
- Every machine must follow the fixed order of jobs. In the case of the re-entrant machine, if the setup times have been met, the machine has the choice of which operation to start, while still following the jobs' order. For example, $o_{(2,3)}$ cannot be performed before operation $o_{(1,3)}$ has been done.
- Every relative due date must hold. Therefore, if $D_{(o_a,o_b)}$ is defined, then the start of operation b cannot be later than the start time of operation a plus the value of the relative due date.

The optimality of a schedule can be assessed on multiple criteria. The ones we use are minimal makespan and maximal robustness or flexibility. The makespan is equal to the moment when the last operation is finished, while the robustness or flexibility of a solution is defined as the ability of the schedule to deal with unexpected situations.

3 Related Work

This section presents in a general to specific approach the work that has been conducted in studying sensitivity analysis, robustness measures and scheduling problems modelled as re-entrant flow shops.

For many years, sensitivity analysis has been a topic undergoing intense study in different branches of science, like managerial sciences or mathematical sciences. Hall and Posner, in their paper [14], make the first systematic sensitivity analysis for different types of scheduling problems. They base the need for their sensitivity analysis study on the work of Anderson et al. [15], who explain from a managerial perspective that every schedule exists in a continuously changing environment. In other words, they create the hypothesis for introducing uncertainties in scheduling problems to closely mimic real-life situations. Moreover, while explaining an algorithm, Hall and Posner acknowledge the need for new methods to choose the most robust schedule from a pool

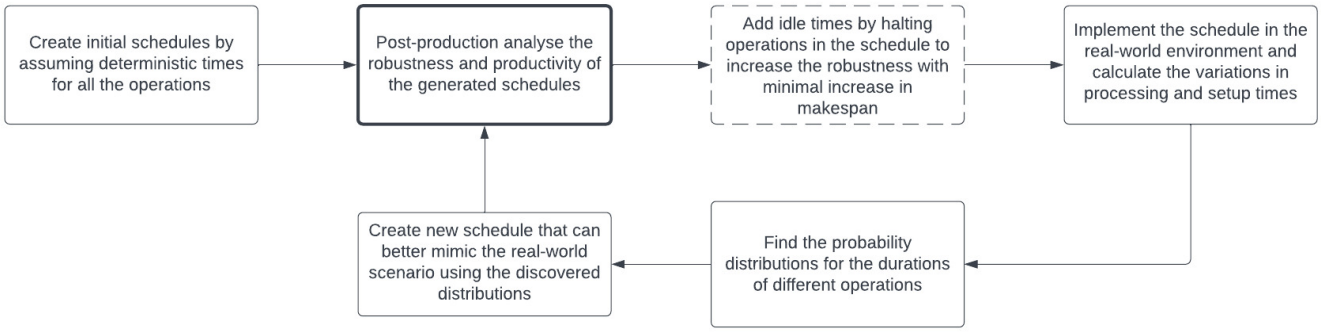


Figure 1: The processes involved in putting a schedule into action in a real-world setting. We are particularly interested in the first run through the second step, which is illustrated with a bold border in this figure. The step with a dotted border is an optional step.

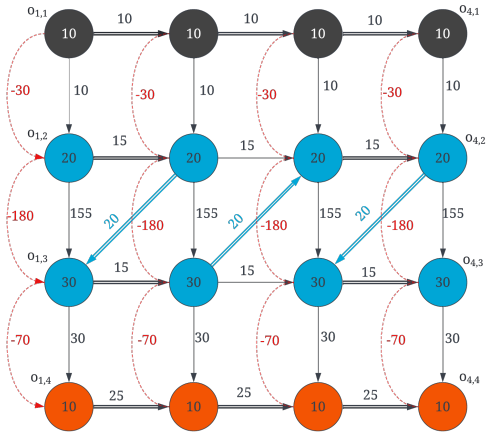


Figure 2: An example of a flow shop with three machines, four jobs and four operations. The operations that have the same colour are performed by the same machine, with the number inside the circles representing the processing times. The setup times are shown with black edges and the relative due dates are presented with red edges. The blue edges represent sequence-dependent setup times. The double edges illustrate a possible solution to this flow shop.

of optimal solutions for scheduling problems. Penz et al. [16] focus on sensitivity analysis for static scheduling algorithms and on determining the performance guarantee of a solution in case of disturbances. They clearly make the distinction between sensitivity analysis, robustness and stabilization process. Al-Fawzan and Haouari [9], while investigating a project scheduling problem, define robustness. In their paper, Al-Fawzan and Haouari define robustness in a deterministic way. Other papers, like the one from Shen et al., [17] define robustness with a probabilistic approach.

In our case, the scheduling problem has been created by the needs of an industrial partner. Therefore, uncertainties have to be taken into account and analysed, following the principles described by Anderson et al. in their work [15]. Then, Pixten et al. [12] formally define the scheduling problems modelled as re-entrant flow shops. This formalisation has every constraint needed in our problem and it gives us the possibility to apply algorithms for graphs to solve this problem. Moreover, Waqas et. al [4], having a similar use case for their 2 re-entrant flow shops, help us visualise the problem in a real-life situation with industrial printers. Tempel et al. [18] propose a heuristic, MPHCS, which takes into account the relative weights for productivity and flexibility that influence how the schedule is created. They claim that for 50% of their test cases, the algorithm finds a schedule within 10% of the optimal makespan. However, it is not clear what weights they are using for the experiments and how the variation in the weights affects the results.

To the best of our knowledge, most of the existing literature on the topic uses robustness measurements as a way to create initial robust schedules that can accommodate unexpected disturbances. However, the literature lacks an exhaustive study for the most optimal robustness measures for different scheduling problems, containing various constraints and degrees of freedom for scheduling solutions. Al-Fawzan and Haouari [9] describe robustness as the sum of free time slack for all the operations in the schedule. Then, Shen et al. [17] come up with two different stochastic ways of measuring the robustness of schedules. Since the literature lacks an exhaustive study for the most optimal robustness measures for different scheduling problems, it was impossible to find a suitable measurement for analysing the robustness of our schedules and comparing schedules that prefer flexibility to productivity. Branke and Mattfeld [10] introduce a flexibility term that penalizes idle times. The results obtained by using this measure were consistent, but we believe the underlying idea is not extendable to scheduling problems where idle time insertion is beneficial. Therefore, we want to analyse the robustness of the schedules created for industrial printers and create the means for future work in robust initial scheduling. Our problem is NP-Hard, though the only proofs that have been published are related to similar but not exactly the same problem [19].

4 Robustness Analysis

This section explains our approach to bound the amount of time slack a schedule made for a system of difference constraints has before it becomes infeasible. Furthermore, we define a schedule-specific robustness measure to compare dif-

ferent solutions for the same scheduling problem.

4.1 Bounded Intervals of Time Slack

Now, to illustrate the described behaviour, we consider a simplified example. Suppose we have a solution for a scheduling problem with 3 machines and 5 jobs and we want to measure how much disturbance the solution can absorb before it breaks down. Assume that an unexpected delay happens during the first operations from the first two jobs and the solution remains feasible. Now, assume the same disturbance happens for the second operations from the same jobs. Just because the schedule was able to deal with the first delay, it is not guaranteed that any other delay will not break the schedule. Therefore, we quickly come to understand the answer to our question is not a number, but an interval.

In case of a delay, a solution can have various behaviours. The first and most convenient case is that the solution can absorb the delay without any increase in starting times for any of the subsequent operations and consequently no increase in the total makespan of the schedules. However, this can only be the case for systems that have idle times between the finish and start times of subsequent operations. The second case is when a delay can only be absorbed by increasing the start times of some or all the subsequent operations, to have time for setting up the machines. The third case happens when increasing the start times is not enough and rescheduling is needed to solve the scheduling problem feasibly. The fourth and worst case happens when a delay makes it impossible to reschedule the solution and the whole scheduling problem becomes unsolvable. In this paper, since we are interested in analysing the robustness of schedules, and not scheduling problems, we only consider and study the first three cases.

After presenting the possible outcomes of higher than expected processing times, we can formally define the interval of delay that can be absorbed by a schedule. We know that in our problem, the due dates are relative, meaning that they enforce a maximum period between the start of two operations. This led to the definition of the lower bound of our interval. We defined the lower bound as the minimum difference between the due date for the start of an operation and the processing and setup times of the operations that happen between the operations that characterise the due date, for any two operations in the schedule. This is the lower bound of slack time for a specific schedule. In other words, a disturbance has to be strictly greater than this number for it to have a chance to make the schedule infeasible. Formally, it was defined as:

$$Min\left(D\left(_{(j,o-1),(j,o)}\right) - \sum_{b\,\in\,B} P\left(_{j,b}\right) - \sum_{b'\,\in\,B'} S\left(_{j,b'}\right)\right) \tag{2}$$

for all D in the set of defined due dates, where B defines all the operations that happen between operation $o-1$, included, and operation o. B' defines all the operations that happen between operation $o-1$ and operation o, included.

To define the upper bound of this interval, we had to see how much disturbance the schedule can absorb before it becomes for sure infeasible. The formula for the lower bound could be adapted to represent the upper bound of our interval.

Algorithm 1 Calculate the Problem-Specific Robustness

```
dueDatesSum ← 0
robustness ← 0
repeat
    duration ← startOp − startPrevOp
    slack ← dueDate − duration
    robustness ← robustness + slack
    dueDatesSum ← dueDatesSum + dueDate
until all the defined due dates
return robustness/dueDatesSum
```

Instead of looking for the minimum difference, we were interested in adding this difference for every relative due date.

$$\sum_{o\in O}\left(D\left(_{(j,o-1),(j,o)}\right) - \sum_{b\,\in\,B} P\left(_{j,b}\right) - \sum_{b'\,\in\,B'} S\left(_{j,b'}\right)\right) \tag{3}$$

Therefore, in case a disturbance happens in a specific place, the schedule has a time slack of as little as described in (2). Otherwise, if the disturbance is evenly distributed, it can have as much time slack as defined in (3).

It is important to note that different scheduling problems might have a different number of defined due dates. Therefore, two schedules created for two different scheduling problems cannot be compared using this interval.

4.2 Problem-Specific Robustness Measure

The interval described in Section 4.1 can give a clear indication of how robust a single schedule created for a specific scheduling problem is. However, it has limitations when it comes to comparing two different schedules for the same problem.

To have a relative ordering of schedules in terms of robustness, we created a new measure formula. To come up with this formula, we needed a base that remains unchanged for any solution for the same problem. We found that no matter the chosen path, the relative due dates remain the same. Moreover, when encountering delays, we applied several rescheduling algorithms like Bellman-Ford or one of its variations, called Goldberg Radzik [20]. We discovered that depending on the chosen path the distance of every operation to its deadline will change. Therefore, we used this distance in our measurement and normalised it by dividing it by the total sum of relative deadlines in the schedule. Formally, it was defined for all the relative due dates, as:

$$\frac{\sum\left(D_{((j,o-1),(j,o))} - \left(Start_{(j,o)} - Start_{(j,o-1)}\right)\right)}{\sum D_{((j,o-1),(j,o))}} \tag{4}$$

This robustness measure can be calculated in linear time in terms of defined due dates for any schedule and it can be used in further research for initial robust scheduling with idle time or rescheduling in case of deterioration for systems defined as flow shops with relative due dates. The implementation can be seen in Algorithm 1.

5 Experimental Setup and Results

The experimental results aimed to assess the quality of our robustness measure by comparing it with a measure created by Branke and Mattfeld [10]. Their paper introduced this measure to decrease the nervousness of schedules, defined by them as the disruption of the schedule in case of re-scheduling, and increase the schedules' performance over time. They called it *flexibility term* and it was denoted by *P*. Even if it has a different name than ours, it ought to achieve the same result, increased robustness for dynamic schedules. In this section we present the experimental setup followed by the experimental results, that should answer the following questions:

- Is our problem-specific robustness measure effective in assessing the flexibility of different schedules?
- Is our measure consistent when it comes to multiple scenarios?
- How does our measure compare to the state-of-the-art measure?
- Can we systematically assess which is the most robust schedule for a given benchmark scheduling problem?

5.1 Experimental Setup

To conduct these experiments, we used scheduling problems from an industrial partner. The 3-machine, 2-re-entrant scheduling problems represented the benchmarks. They consisted of 100 and 500 jobs, each of them with 4 operations. The processing times, due dates and setup times were represented using integer values.

For each benchmark, three schedules were generated using the MPHCS algorithm [18]. The algorithm has three parameters P (productivity), F (flexibility), T (tie-breaker), which can be assigned different relative weights that indicate the importance of productivity and flexibility in the schedule. For this experiment, we generated three different types of schedules: *a*, *b*, and *c* with different weights.

- *a*: $P = 0.7$, $F = 0.25$, $T = 0.05$
- *b*: $P = 0.45$, $F = 0.45$, $T = 0.1$
- *c*: $P = 0.25$, $F = 0.7$, $T = 0.05$

Even though there exists a flexibility term in the algorithm, its behaviour related to our robustness measure is unpredictable for multiple reasons. The first reason is that they define flexibility in a different way than us. According to their definition, the more flexible a schedule is, the more feasible possibilities are available for planning following jobs. The second reason is the lack of linearity in the scheduling behaviour in relationship with the given parameters. To better understand how the parameters affect the solution we generated 50 schedules for each relative weight of the flexibility in the intervals 0.05 and 0.85, with an increment of 0.05. Therefore we analysed 850 schedules generated for the same benchmark. We found out that there was no change in the schedule when using relative weights for F between 0.05 and 0.35. The first change was visible at 0.40 and lasted until 0.75, included. Even though the weight of the flexibility increased, the median robustness calculated using our measure decreased. Then, at 0.8 there was a trade-off between productivity and robustness that resulted in a gain in robustness but a loss of productivity, with the makespan greatly increasing. This behaviour was summarized in Figure 3.

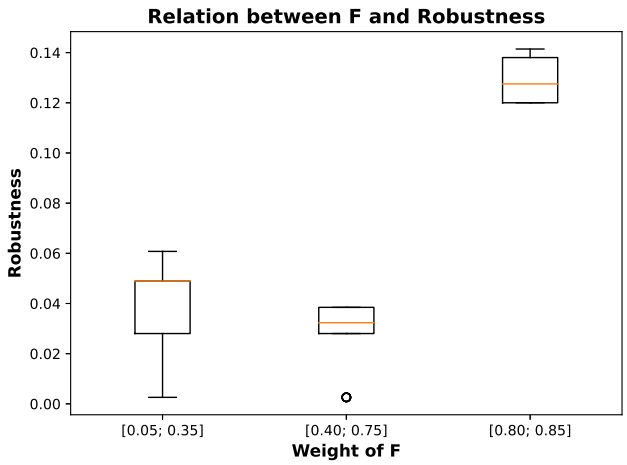


Figure 3: Box plots that illustrate the absence of linearity in robustness for schedules generated using the MPHCS heuristic [18].

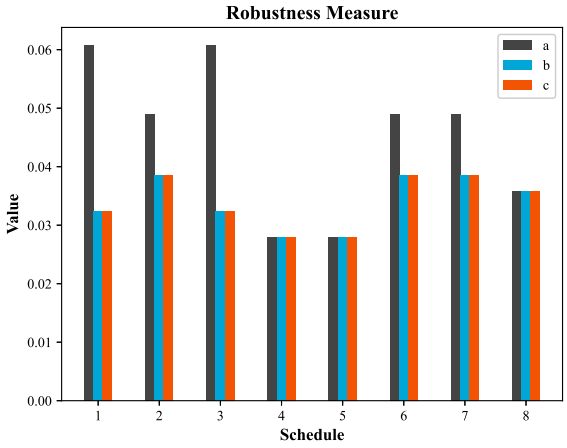
The solutions had different makespans and execution sequences for the re-entrant machines while following the same constraints. The model ran on the Gurobi [21] server using an academic license. The generated schedules were in a text file format and contained the starting times of all the operations in the solution. The rows represented jobs and the columns represented operations. Since the start times were already available, it could be considered that this was a post-production analysis of the generated schedules.

The experiments were performed in an environment containing Python 3.9, on an Intel Core i7 CPU running 2.8GHz with Windows 10.

5.2 Experimental Results

To assess the validity of our robustness measure, we analysed some of the schedules that were generated as explained above. The schedules were known to be feasible, so no validation had to be done. Then, the robustness values for each of the eight schedules from the three categories were calculated using the implementation described in Algorithm 1.

After running the experiments for our robustness measure, we aggregated the results in Figure 4. The y-axis shows the robustness value yielded by our measure and the x-axis presents the benchmark for which the schedules were generated. In general, the robustness value was in an interval between approximately 0.03 and 0.06. To contextualise these numbers, the ideal value of one would have meant that all the operations and their setup times finished instantly and no time was spent, which was impossible. At the opposite end of the spectrum, a value of zero would have meant that any unexpected delay would break the system and re-scheduling was needed, which is an expensive operation in terms of time. We saw that the schedules in *a* had the highest value every time.



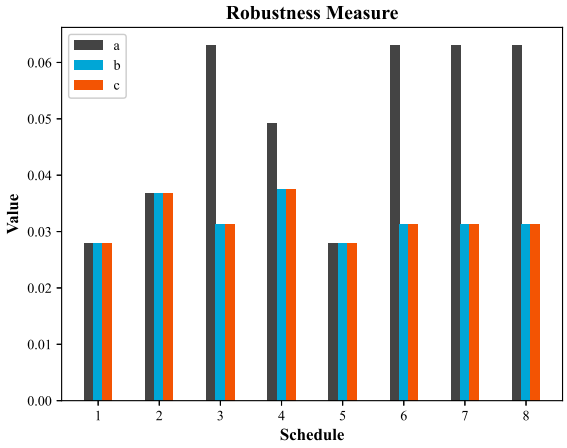
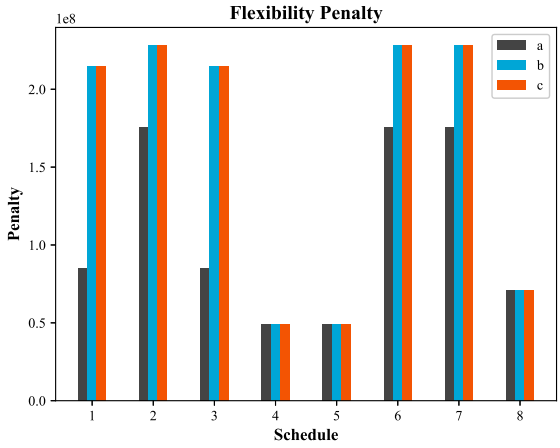


Figure 4: Comparison of different solutions for benchmarks with 100 jobs (top) and 500 jobs (bottom). The y-axis represents the value calculated using the formula 4, the x-axis shows the benchmark and *a,b and c* show three types of solutions.



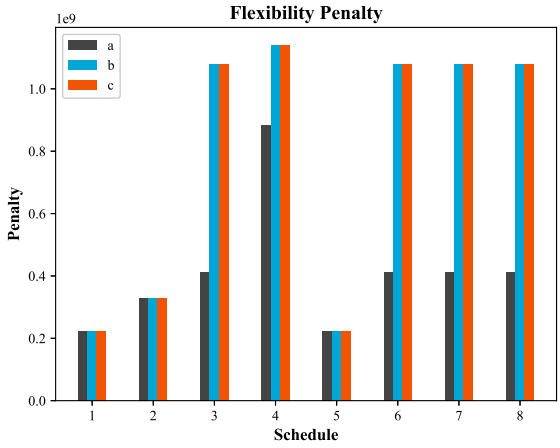


Figure 5: Comparison of different solutions for benchmarks with 100 jobs (top) and 500 jobs (bottom). The y-axis represents the penalty calculated using the flexibility formula from the paper [10], the x-axis shows the benchmark and *a,b and c* show three types of solutions.

This meant that the schedules were more robust or flexible than those in *b* or *c*. Therefore, in case of processing times or setup times that took longer than expected, the schedules in *a* had the highest probability to accommodate the delay.

To further assess the quality of our robustness measure, we implemented a state-of-the-art flexibility measure, as described in [10]. The paper explains that because early idle periods are expected to be particularly important for dynamic scheduling, the penalty for each idle time is weighted linearly decreasing with time, as:

$$w(t) = max\left\{0, 1 - \frac{t}{\beta}\right\}$$

Since this measure penalized the idle times, all the schedules that we applied this formula to had the common characteristic of starting every operation as soon as possible. This restriction might not always be beneficial for the real world, since halting the start of an early job might increase the robustness of the schedule without even increasing the total make-span. Our robustness measure does not suffer from this caveat and can therefore be generalised for a wide range of dynamic scheduling problems. However, it was beneficial to make the comparison to prove our measure's accuracy.

In order to perform the comparison, we deemed it appropriate to choose the value for β equal to the finishing time of the last operation to take the whole time interval into account. Moreover, we came up with an implementation in Python, as described in Algorithm 2. After calculating the values for the same benchmarks and the same schedule types as in Figure 4, we aggregated them in Figure 5. We observed that both of the measures yielded similar results and our measure could be scaled to schedules with 500 jobs without any unwanted value increase. The schedules in *a* obtained the lowest penalty each time, which makes them the most flexible schedules out of the three. Then, we saw that the penalties for the schedules in *b* and *c* were equal. Again, the efficiency of our measure was confirmed. Moreover, the difference in penalty between schedule *a* and schedules *b* and *c* had a similar percentage as the difference in their robustness values. However, their flex-

Algorithm 2 Calculate the Flexibility Penalty

$beta \leftarrow lastOperationStart + processingTime$
$penalty \leftarrow 0$
repeat ▷ Find operations' sequence on machine
  repeat
    $t \leftarrow opStart + processingOp + setupNextOp$
    $w \leftarrow 1 - \frac{t}{\beta}$
    $idle \leftarrow startNextOp - t$
    $penalty \leftarrow penalty + (idle * w)$
  until all the operations on the machine
until all the machines
return $penalty$

ibility measure classified schedules four and five as the most robust or flexible schedules. On the other hand, our robustness measure assigned the most robustness to schedule *a* for the benchmarks one and three. The explanation for this is that there was more idle time in the beginning of the schedules *a* for the benchmarks one and three, which we considered an advantage since the machines have an equal probability of breaking down or taking longer than expected.

6 Responsible Research

In order to show that we have followed the principles of Responsible Research, we talk about our transparency regarding the software used, the randomly chosen test set and the reproducibility of our results. Firstly, in order to generate the schedules for the benchmarks from the industrial partner, we used the Gurobi [21] server with an academic license. Secondly, in order to test our robustness measure and compare it with another known technique, we used a subset from the generated solutions. To not interfere with the accuracy of the results, the first eight schedules were selected for each type: *a, b and c*. Lastly, the reproducibility of our results is high, as the algorithms are presented in pseudocode in the paper and the test set can be made available at request.

7 Conclusions and Future Work

We have studied the re-entrant flow shop problem with processing times, relative due dates and sequence-dependent setup times expressed as integer values. First, we have formally defined the industrial printer as our use case. Then, we found how much time a schedule has before it becomes infeasible by bounding it within an interval containing its lower and upper bounds. Moreover, we empirically developed a new robustness measure. We applied it to a set of schedules from our industrial partner. Then, we implemented a known flexibility measure to assess its validity and applied it to the same schedule set. It confirmed that our robustness measure is as accurate as the previously known technique while having the ability to be used in flow shops with idle time inserted, as opposed to the known measure. For future studies, initial scheduling for re-entrant flow shops should take into account the possibility of inserting idle times in designated places to improve the robustness of a schedule, with minimal impact on the duration. This study can be done using our developed measure to understand the impact of halting some operations in any system represented as a flow shop with relative due dates.

References

[1] B. Çaliş and S. Bulkan, "A research survey: review of ai solution strategies of job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 26, no. 5, pp. 961–973, 2015.

[2] M. Zweben and A. R. Center., *Constraint-based scheduling [microform] / Monte Zweben*. NASA, Ames Research Center, Artificial Intelligence Research Branch ; National Technical Information Service, distributor [Moffett Field, CA] : [Springfield, Va], 1991.

[3] J. Hoogeveen, J. Lenstra, and B. Veltman, "Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard," *European Journal of Operational Research*, vol. 89, no. 1, pp. 172–175, 1996.

[4] U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. Corporaal, "A reentrant flowshop heuristic for online scheduling of the paper path in a large scale printer," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 573–578, 2015.

[5] M. B. Do, W. Ruml, and R. Zhou, "On-line planning and scheduling: An application to controlling modular printers," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, p. 1519–1523, AAAI Press, 2008.

[6] C.-L. Chen, V. S. Vempati, and N. Aljaber, "An application of genetic algorithms for flow shop problems," *European Journal of Operational Research*, vol. 80, no. 2, pp. 389–396, 1995.

[7] K. R. Baker and D. Altheimer, "Heuristic solution methods for the stochastic flow shop problem," *European Journal of Operational Research*, vol. 216, no. 1, pp. 172–177, 2012.

[8] J. Xiong, L. ning Xing, and Y. wu Chen, "Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns," *International Journal of Production Economics*, vol. 141, no. 1, pp. 112–126, 2013. Meta-heuristics for manufacturing scheduling and logistics problems.

[9] M. A. Al-Fawzan and M. Haouari, "A bi-objective model for robust resource-constrained project scheduling," *International Journal of Production Economics*, vol. 96, pp. 175–187, 5 2005.

[10] J. Branke and D. C. Mattfeld, "Anticipation and flexibility in dynamic scheduling," *International Journal of Production Research*, vol. 43, pp. 3103–3129, 8 2005.

[11] T. S. Lee and Y. T. Loong, "A review of scheduling problem and resolution methods in flexible flow shop," *International Journal of Industrial Engineering Computations*, 2019.

[12] J. V. Pinxten, U. Waqas, M. Geilen, T. Basten, and L. Somers, "Online scheduling of 2-re-entrant flexible manufacturing systems," *ACM Transactions on Embedded Computing Systems*, vol. 16, 9 2017.

[13] H. Emmons and G. Vairaktarakis, *Flow shop scheduling: theoretical results, algorithms, and applications*, vol. 182. Berlin, Germany: Springer Science & Business Media, 2012.

[14] N. G. Hall and M. E. Posner, "Sensitivity analysis for scheduling problems," *Journal of Scheduling*, vol. 7, pp. 49–83, 2004.

[15] D. Anderson, D. Sweeney, and T. Williams, *Contemporary Management Science with Spreadsheets*. Mason OH: South-Western College Pub, 1 ed., 1998.

[16] B. Penz, C. Rapine, and D. Trystram, "Sensitivity analysis of scheduling algorithms," *Eur. J. Oper. Res.*, vol. 134, pp. 606–615, 2001.

[17] X. N. Shen, Y. Han, and J. Z. Fu, "Robustness measures and robust scheduling for multi-objective stochastic flexible job shop scheduling problems," *Soft Computing*, vol. 21, pp. 6531–6554, 11 2017.

[18] R. van der Tempel, J. van Pinxten, M. Geilen, and U. Waqas, "A heuristic for variable re-entrant scheduling problems," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, pp. 336–341, 2018.

[19] A. Pishevar and R. Tavakkoi-Moghaddam, "β - robust parallel machine scheduling with uncertain durations," *Universal Journal of Industrial and Business Management*, vol. 2, pp. 69–74, 3 2014.

[20] A. V. Goldberg and T. Radzik, "A heuristic improvement of the bellman-ford algorithm," *Applied Mathematics Letters*, vol. 6, no. 3, pp. 3–6, 1993.

[21] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual." Available at https://www.gurobi.com, 06 2022.