A Re-entrant Flowshop Heuristic for Online Scheduling of the Paper Path in a Large Scale Printer

Umar Waqas*, Marc Geilen*, Jack Kandelaars†, Lou Somers†, Twan Basten*,
Sander Stuijk*, Patrick Vestjens†, Henk Corporaal*
* Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands.
† Research and Development, Océ Technologies, Venlo, The Netherlands.
Corresponding author email: u.waqas@tue.nl

***Abstract*—A Large Scale Printer (LSP) is a Cyber Physical System (CPS) printing thousands of sheets per day with high quality. The print requests arrive at run-time requiring online scheduling. We capture the LSP scheduling problem as online scheduling of re-entrant flowshops with sequence dependent setup times and relative due dates with makespan minimization as the scheduling criterion. Exhaustive approaches like Mixed Integer Programming can be used, but they are compute intensive and not suited for online use. We present a novel heuristic for scheduling of LSPs that on average requires 0.3 seconds per sheet to find schedules for industrial test cases. We compare the schedules to lower bounds, to schedules generated by the current scheduler and schedules generated by a modified version of the classical NEH (MNEH) heuristic [1], [2]. On average, the proposed heuristic generates schedules that are 40% shorter than the current scheduler, have an average difference of 25% compared to the estimated lower bounds and generates schedules with less than 67% of the makespan of schedules generated by the MNEH heuristic.**

I. Introduction

Many Cyber Physical Systems (CPS) require scheduling of events in the system. For example when a silicon wafer is processed in a wafer fabrication, or when a Printed Circuit Board (PCB) is printed in a manufacturing unit or when a sheet is printed in a Large Scale Printer (LSP). A LSP prints thousands of sheets per day at high quality for books, commercial letters etc. The total time spent to print the sheets determines the productivity of a LSP. Increasing the productivity of a LSP is a commercial concern. This paper describes the problem of scheduling sheets in the paper path of a LSP and provides a heuristic to increase the productivity of a LSP.

A paper path in a printer is the path that sheets follow through different components. Figure 1 shows a paper path of a duplex printer (i.e. a printer that prints on both sides of a sheet). The arrows represent the flow of sheets and a hexagon is a component that processes sheets. A sheet enters from the Paper Input (PI) component, gets printed on its first side in the Image Transfer (IT) component (referred to as the first pass), gets turned upside down by the Turn (T) component and returns back to get its second side printed (the second pass). After the second pass a sheet leaves through the Finish (F) component. At the *merge point* the return path meets the input path and at this point a scheduling decision is made whether to first print a returning sheet or a new sheet. This decision is referred to as determining the *interleaving* of sheets. The second pass sheets re-enter the print section and are called

This work was supported in part by the Dutch Technology Foundation STW, project NEST 10346.

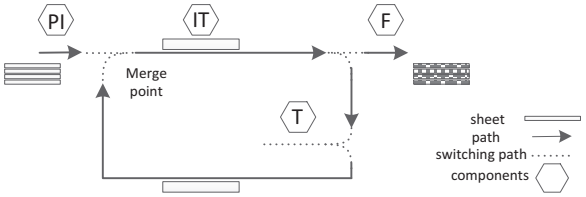


Fig. 1. Flow of sheets through the paper path in a LSP.

re-entrant sheets. We assume for simplicity that all sheets are duplex sheets. Under this assumption, simplex sheets can still be printed by allowing the simplex sheets to re-enter but not print them for the second time.

The arrangement of components in a paper path is similar to arrangement of machines in *shop* scheduling [3]. The components are shared between the sheets. The IT component may need to perform additional steps, called *setups*, before printing the next sheet. For example, printing on a coated paper may require to heat up the ink. If the ink was already at the required temperature (because the previous sheet required the same temperature) then no setup is required. Thus it depends on interleaving decisions whether a setup is required or not. This conditional setup is known as *sequence dependent setup* [3]. In practice, the setup times are significantly larger (10-15 times) than the print times. The heuristic consecutively interleaves sheets with similar setup requirements to minimize the number of setups and to increase the productivity of a LSP.

The re-entrant sheets flow back to the merge point over the return path. There are bounds on the speed it can travel on the return path. A *relative due date* is the resulting upper bound between the first and second passes of a sheet. The combination of re-entrance, sequence dependent setup times and relative due dates define the specific scheduling problem. It is compute intensive to solve the scheduling problem to optimality. The re-entrant flowshop with setup times and relative due dates has not been studied before.

The following section defines the re-entrant flowshop scheduling problem. In Section III *constraint graphs* are introduced that are used by the heuristic approach described in Section IV to determine schedules. Section V describes the experiments performed to evaluate the proposed heuristic. Section VI describes the related work. Section VII concludes the paper.

II. Problem definition

A LSP consists of components that perform operations on sheets. The operations we consider are: input a sheet, print a sheet and finish a sheet. A LSP is a special case of re-entrant flowshop [2], [3]. The re-entrant flowshop

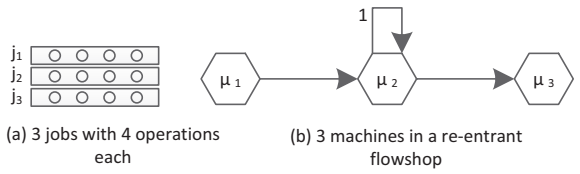


Fig. 2. The jobs (a) and the re-entrant machines (b) in a LSP.

studied in this paper is defined in Definition 1 and Figure 2 shows an example.

Definition 1. A *re-entrant flowshop with sequence dependent setup times and relative due dates* is a tuple $(J, r, O, P, S, D, M, \phi)$ defined as follows. $J = \{j_1, \ldots, j_n\}$ is the set of jobs. In Figure 2(a) a job is a sheet in the LSP. r is a positive integer that denotes the number of operations performed on each job. For the LSP in this paper $r = 4$ i.e., every job has 4 operations. O is the set of operations performed on jobs. The circles in Figure 2(a) in a job are operations. Let $O_i = \{o_{i,1}, \ldots, o_{i,r}\}$ be the set of r operations performed on job j_i, then $O = O_1 \cup \cdots \cup O_n$. The function $P : O \to \mathbb{N}$ determines the processing times for operations in O. $S : O \times O \to \mathbb{N}$ describes the setup time between operations. $S(o_x, o_y)$ is the time that a machine needs after the execution of o_x ends to prepare for the start of execution of o_y. $D : O \times O \to \mathbb{N}$ describes the due dates between operations. Let $start(o)$ be the time an operation o starts. A due date between operations o_x and o_y denotes that o_y must start no later than $D(o_x, o_y)$ time units after o_x has started. i.e. $start(o_y) \leq start(o_x) + D(o_x, o_y)$. $M = \{\mu_1, \ldots, \mu_m\}$ is the set of machines. The LSP has 3 machines (components). In Figure 2(b) arrows show the flow of jobs and machine μ_2 is a re-entrant machine. ϕ is a vector of machine numbers that defines the re-entrance pattern of jobs through machines. $\phi =< \gamma_1, \ldots, \gamma_r >$, $\gamma_i \in \{1, \ldots, m\}$. The re-entrance vector for the LSP is $< 1, 2, 2, 3 >$.

We assume that a re-entrant flowshop has the following properties. Machines are ordered as $\mu_1, \mu_2, \ldots, \mu_m$. The first operation is performed by μ_1. Depending on the re-entrance vector a job either re-enters the same machine or moves to the next machine. All jobs follow the same order of machines and the jobs share the machines. Operations are non-preemptive and machines can process at most one operation at a time. A machine is available immediately after completion of an operation. The job order is fixed and is $j_1, \ldots, j_n$. Following is an example of a re-entrant flowshop with a single machine and 3 jobs.

Example 1. Let a re-entrant flowshop consist of a machine $M = \{\mu_1\}$, three jobs $J = \{j_1, j_2, j_3\}$ and $r = 2$ operations performed on a job. The operations in a job j_i are $O_i = \{o_{i,1}, o_{i,2}\}$. The processing times for all operations are equal to 1. The due dates are $D(o_{2,1}, o_{2,2}) = 7$ and ∞ (indicating no due date) for any other combination of operations. The re-entrance vector is $\phi =< 1, 1 >$. The setup times between operations are in the matrix S'.

$$S' = \begin{array}{c} \begin{matrix} o_{1,1} & o_{1,2} & o_{2,1} & o_{2,2} & o_{3,1} & o_{3,1} \end{matrix} \\ \left[\begin{array}{cccccc} 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 \\ 0 & 0 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} o_{1,1} \\ o_{1,2} \\ o_{2,1} \\ o_{2,2} \\ o_{3,1} \\ o_{3,2} \end{matrix}$$

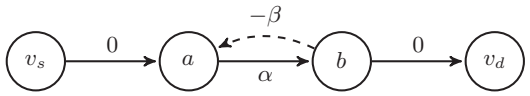


Fig. 3. An example of a constraint graph.

A solution to a reentrant flowshop problem is a schedule, defined as follows.

Definition 2. A *schedule* $\Pi : O \to \mathbb{N}$ describes the start time of every operation and respects: the job and operation ordering, exclusive use of machines, setup times and due dates. The *makespan* σ_Π of a schedule Π is the total duration of the schedule. σ_Π is the completion time of the last operation in the schedule i.e. $\sigma_\Pi = \Pi(o_{n,r}) + P(o_{n,r})$.

Note that $o_{n,r}$ is the last operation to finish in a schedule due to the fixed job and operation order. The scheduling criterion for a re-entrant flowshop is to find a schedule Π_s having a minimal makespan: $\underset{\Pi_s}{\arg\min}\, \sigma_{\Pi_s}$.

A valid schedule always exists for the LSP considered in this paper. An intuitive argument is that a single job always has a schedule and that a schedule for a set of jobs can always be formed by considering each job separately without interleaving. Although a non-interleaving schedule always exists, it will not be as productive as a schedule with interleaving because travelling back on the return path takes more time than interleaved printing. Several re-entrant flowshops without fixed job ordering have been shown to be NP-Complete [3]. Proving the complexity of the re-entrant flowshop problem with fixed job orderings is left as a future work.

The start times of operations are constrained by setup times, processing times and relative due dates. In Section III a graph based model is introduced to uniformly model the constraints in a re-entrant flowshop.

III. Constraint graph model

The *constraint graph* models the constraints between operations in a re-entrant flowshop.

Definition 3. A *constraint graph* is a tuple $cg(V, E = E_C \cup E_O, w, G, g, v_s, v_d)$. V is a set of vertices. Vertices in a constraint graph denote *events*, for instance a and b in Figure 3. $E \subseteq V \times V$ is a set of directed edges. An edge $(a, b) \in E$ with weight $w(a, b)$ denotes the minimum distance constraint between events a and b. Let t_a and t_b be the start times of a and b respectively then the distance constraint is $t_b \geq t_a + w(a, b)$, with $w : E \to \mathbb{Z}$. In Figure 3 the weight $w(a, b) = \alpha$ and the edge (b, a) has a constraint $t_a \geq t_b - \beta$. The edge set E is partitioned into two sets E_C and E_O. The set E_C consists of *compulsory constraints* that must never be violated. The constraint graph with only its compulsory edges is called *compulsory constraint graph*. Solid edges in Figure 3 are compulsory constraints. The set E_O contains *optional constraints* that can be added to E_C to enforce additional constraints if needed to compute a schedule. Dotted edges in Figure 3 are optional constraints. $G : V \to \{1, \ldots, g\}$ determines the group number of a vertex in V. There are total $g \in \mathbb{N}$ groups. A group is used to model vertices of events that share a resource. The source vertex $v_s \in V$ is a vertex with no incoming edge. The destination vertex $v_d \in V$ is a vertex with no outgoing edge. The compulsory constraint graph is fully connected.

The time instant at which an event occurs is called the *occurrence time* of the event. It is defined as follows.

Definition 4. A *occurrence time* of vertex $a \in V$ is $t_a \in \mathbb{N}$ such that all compulsory constraints hold. Schedule $T : V \rightarrow \mathbb{N}$ is a mapping of an event to an occurrence time.

A function T is a feasible schedule of a constraint graph if and only if all compulsory constraints hold on the schedule. A schedule T can be computed by finding a shortest path of compulsory edges only between the source vertex and the other vertices, for instance using the Bellman-Ford algorithm. Events may require resources at the time of occurrence. For example, printing a sheet requires that the IT component is free. Therefore events in the same group must be *ordered* and constraints ensure that the resources are used exclusively. The vertices are ordered by enforcing edges from E_O. E_O is assumed to contain the constraints for all possible orderings.

Definition 5. A *path* p in a constraint graph is a non-empty sequence of connected edges $(a, b) \rightarrow \cdots \rightarrow (c, d)$ from the set E. The length l_p of path p is the sum of the weights of the edges in p i.e., $l_p = w(a, b) + \cdots + w(c, d)$. The path starts from vertex a and ends at vertex d. A *cycle* is a path starting and ending at the same vertex.

Definition 6. A *non-negative* path p is a path consisting of only edges with non-negative weights i.e., for any edge (a, b) in p its weight $w(a, b) \geq 0$.

In Figure 3 $(v_s, a) \rightarrow (a, b)$ is a non-negative path with length α (assuming $\alpha > 0$). The path $(a, b) \rightarrow (b, a)$ is a cycle with length $\alpha - \beta$. Cycles in the compulsory constraint graph with length greater than 0 cannot be satisfied by schedules and are thus prohibited.

Definition 7. A vertex *a precedes* vertex b if there exists a non-negative path of compulsory edges from E_C starting from a and ending at b. A vertex *a immediately precedes* vertex b if there exists an edge $(a, b) \in E_C$ with non-negative weight $w(a, b) \geq 0$.

Choices in interleaving of jobs are different possibilities to order a set of vertices. An ordering tuple represents one such choice.

Definition 8. A tuple $ot((a, b), (b, c))$ of edges (a, b) and (b, c) from E is called an *ordering tuple* for vertex b if a immediately precedes c. An ordering tuple is *feasible* if and only if adding the edges in the tuple to E_C does not introduce a cycle with positive length in the compulsory constraint graph.

We assume that E contains all edges for an ordering tuple for all possible interleavings. The set of ordering tuples for vertices in the groups is defined as follows.

Definition 9. The set $OT_b^x = \{((a, b), (b, c)) \mid a, c \in V_x\}$ is the set of all feasible ordering tuples for vertex b in its group x. $OT^x = \bigcup_{b \in V_x} OT_b^x$ is the set of all feasible orderings of all vertices in V_x. $OT = \bigcup_{x \in \{1,\ldots,g\}} OT^x$ is the set of all feasible ordering tuples for all groups.

Determining an optimal ordering of vertices is the *vertex ordering problem* with the following problem statement.

Definition 10. *(Vertex Ordering Problem)* Find a feasible set of ordering tuples $OT^{sol} \subseteq OT$ such that vertices in all vertex groups are totally ordered i.e., For every x in $\{1, \ldots, g\}$: $(\preceq, V_x)$ is a total order and the time of occurrence t_{v_d} of the destination vertex v_d is minimal.

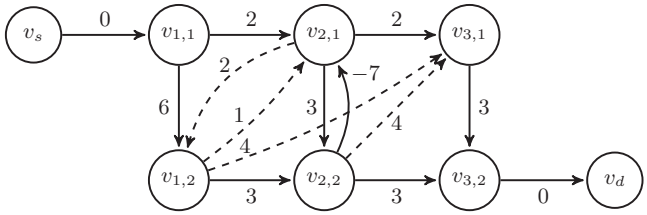


Fig. 4. A constraint graph of the re-entrant flowshop in Example 1.

Figure 4 shows a constraint graph for the re-entrant flowshop of Example 1. The graph has 6 vertices (3 jobs having 2 operations each). There is one group, $g = 1$. The solid edges represent the fixed job and operation ordering constraints. The dotted edges represent optional constraints. The weights of the edges are the sum of the processing times and the setup times. The edge from $v_{2,2}$ to $v_{2,1}$ represents the due date constraint.

IV. Proposed heuristic approach

A schedule for a re-entrant flowshop is the occurrence times of all totally ordered vertices in all groups. The makespan of a schedule depends on the selection of ordering tuples. We present three metrics, namely, *productivity*, *flexibility* and *distance* to assess the impact of an ordering tuple on the constraints and on the makespan. Using these metrics, the heuristic ranks and selects ordering tuples to compute a schedule.

Productivity quantifies the impact of an ordering tuple on the makespan of a schedule. To compute productivity, Equation 1 first computes $d_{x,b}$, which is the maximum possible increase in time of vertex c that can result from any ordering tuple $ot((a, b), (b, c))$ in the set OT_b^x.

$$d_{x,b} = \max_{ot \in OT_b^x} \max(t_c, \max(t_b, t_a + w(a, b)) + w(b, c)) \quad (1)$$

Intuitively, productivity quantifies how restrictive the constraints in the ordering tuple are compared to all possible ordering tuples for a vertex. The ordering tuples that require setups have larger weights $w(a, b)$ and $w(b, c)$ than the ordering tuples not requiring a setup. Hence, $P_{ot} = 0$ indicates a maximally productive ordering tuple. The productivity of an ordering tuple $ot((a, b), (b, c))$ in a set OT_b^x for vertex b is computed in Equation 2.

$$P_{ot} = \frac{\max(t_c, \max(t_b, t_a + w(a, b)) + w(b, c))}{d_{x,b}} \quad (2)$$

An ordering tuple enforces new constraints consuming the time between the due date and the start time of a vertex. Flexibility of an ordering tuple $ot((a, b), (b, c))$ quantifies the effect of the edges over the due date constraints of the vertex b. Let D_b contain all edges with negative weight (due dates) originating from vertex b, then Equation 3 computes the flexibility of the tuple ot.

$$F_{ot} = \min_{(b,x) \in D_b} \frac{t_b - t_x - w(a, b)}{-w(b, x) - w(x, b)} \quad (3)$$

Equation 3 denotes the ratio between the excess amount of time used by an ordering tuple and the total allowed time by the due date constraint. t_b and t_x are the occurrence

times of vertices x and b computed from the partial solution (by considering the edges in E_C and the set OT^{sol}). The fewer the excess usage the more events are feasible if later on the current event is rescheduled. $F_{ot} = 0$ indicates the most flexible ordering tuple.

Let $dist_c$ be the least number of edges required on a path from v_s to vertex c. The distance for the ordering tuple $ot((a,b),(b,c))$ denotes length of the interleaving and is computed as follows.

$$DT_{ot} = \frac{|V| - dist_c}{|V|} \tag{4}$$

Using the three metrics a rank (lower is better) is computed for ordering tuples in Equation 5.

$$R_{ot} = \kappa_P \times P_{ot} + \kappa_F \times F_{ot} + \kappa_{DT} \times DT_{ot} \tag{5}$$

The relative weights (κ_P,κ_F,κ_{DT}) sum up to 1 and indicate the relative importance of the metrics when determining the ordering of events.

Algorithm 1: A heuristic to determine OT^{sol} while minimizing t_{v_d}.

```
1  OT^sol = φ
2  compute vertex groups V_x for x ∈ {1,...,g} using G
3  T = φ
4  for each x ∈ {1,...,g} do
5      while ∃b,c ∈ V_x not ordered in V_x do
6          compute and update T with occurrence
7              times of v ∈ V_x
8          compute set of ordering tuples OT_b^x
9          compute rank R_ot for all ot ∈ OT_b^x
10         select an ot with minimum R_ot
11         add the edges (a,b) and (b,c) from ot to E_C
12         add the tuple ot to OT^sol
13     end
14 end
15 return OT^sol
```

The proposed heuristic approach in Algorithm 1 ranks and picks different ordering tuples to minimize the occurrence time of the destination vertex v_d. The selected ordering tuples are in the set OT^{sol} and denote a feasible schedule. The algorithm starts by computing the vertex groups V_x using the grouping function G. The heuristic iterates for each vertex group and finds a pair of vertices a, b that are not ordered. To order the pair it computes the feasible ordering tuples, ranks them and selects the tuple with minimum rank. The edges in the selected tuple are added to E_C and the tuple is added to OT^{sol}. The algorithm terminates and returns OT^{sol} which is a feasible schedule. The Bellman-Ford algorithm has complexity $O(|V|^3)$ and is used by the heuristic to find whether a schedule is feasible or not. Furthermore, per group, the heuristic iterates with complexity $O(|V|^2)$. Hence, the complexity of the proposed heuristic is $O(g|V|^5)$ where g is the number of groups). The heuristic is evaluated on realistic test cases in Section V.

V. Experimental results

The experiments aim to assess the quality of the heuristic by comparing schedules generated by the scheduler currently in use in our LSP (referred to as the *Eager scheduler*), the Mixed Integer Programming (MIP) scheduler, the modified NEH heuristic from [2] (MNEH) with restricted iterations for online use and estimated lower bounds on the optimal makespan. We present the experimental setup followed by the details of schedulers, the details of the test set and the experimental results.

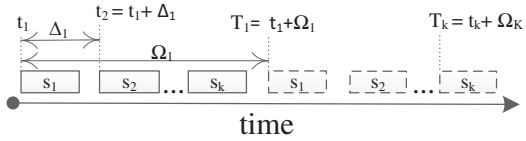


Fig. 5. The operation of the Eager scheduler.

A. Experimental setup

The experiments are performed on two platforms (because of restrictions of the software to different platforms with node locked licenses). The lower bounds, our heuristic scheduler and the MIP were solved using an Intel Core i7 CPU running at 2.67GHz with Ubuntu 10.10. The Eager scheduler and the MNEH [2] were tested on a Intel Core i7 CPU running 3.1GHz with Windows 7. The Eager scheduler is proprietary software and is only available on Windows. However, the differences of the platforms have no effect on the makespans of the schedulers (the main aim of the comparison). The MIP program was solved using CPLEX 12.6, which also provided the lower bounds. CPLEX was run for at most 300 seconds per instance.

B. Details of schedulers, heuristics and lower bounds

The Eager scheduler currently in use in a LSP is a First Fit (greedy) scheduler. Figure 5 shows the computation of a schedule for sheets $s_1, \ldots, s_k$. Let t_1 be the time when sheet s_1 is at the merge point for the first time. The eager scheduler assigns T_1 as the time instant when sheet s_1 will be at the merge point for the second time after travelling on the return path for Ω_1 time units. The first pass of the next sheet, s_2, can be printed Δ_1 time units later than t_1. Δ_1 is the sum of the processing time of s_1 and the setup time between s_1 and s_2. Similarly, the start times, t_i and T_i for $1 \leq i \leq k$ are computed. If the first or the second pass of a sheet cannot be interleaved then the sheet is scheduled at the end of the partial schedule. The simplicitly of the Eager scheduling motivates its use in practice.

The MIPs are solved with CPLEX to compare the heuristic to optimal solutions. However, in many cases, CPLEX failed to find a feasible solution within the time bound. The Lagrangian relaxation of the MIP program is less compute intensive and provides a lower bound (LB) on the makespan of the optimal schedule. Note that the lower bounds do not necessarily have corresponding schedules that achieve a makespan of the lower bound.

Our greedy heuristic is also compared to the state-of-the-art greedy heuristic in literature, for which the MNEH heuristic of [2] is used to generate schedules in only a single iteration. The MNEH heuristic performs a backward iteration over the given initial seed sequence (i.e. a schedule without interleaving). The original MNEH continues to iterate over the (partial) schedule found till no further improvement is possible. On contrast, our heuristic only performs single backward iteration. For fair comparison, the MNEH heuristic was also run in a greedy fashion, i.e., restricted to single backward iteration. In the iteration, the MNEH heuristic starts from the last sheet in the seed sequence and selects the ordering tuple that results in

TABLE I. CATEGORIES AND PATTERNS IN THE TEST SET.

Category	*Pattern*
RA	$(a(b)^+)^+$
RB	$((a \mid aa \mid aaa)(b)^+)^+$
BA	$((c^{10} \mid c^{20}) \mid (d^{10} \mid d^{20}) \mid \cdots \mid (g^{10} \mid g^{20}))^5$
BB	$((h^{10} \mid h^{20}) \mid (i^{10} \mid i^{20}))$
H	$j^+ \mid k^+ \mid \cdots \mid s^+$

minimal makespan for the partial schedule. The heuristic continues to find ordering tuples in backward direction.

C. Test set

The experiments test the heuristic on a set of 700 schedule requests varying in size between 18 and 800 sheets. Sheets in a schedule request can be one of 19 different types. Examples are sheets of different thickness, length etc. The types of sheets with patterns are shown in Table I. These patterns are representatives of typical schedule requests for a LSP. Each pattern is a type of schedule request with a specific pattern of sheets. The patterns are shown as regular expressions with literals $a-s$ where each literal represents a different type of sheet.

The test set consists of 5 categories of patterns. The first category *Repeating A* (RA) consists of cases with *repeating patterns* of sheets in which an a type sheet is followed by several b type sheets. In category *Repeating B* (RB) a test case is one, two or three a type sheets followed by several b type sheets. In category *block A* (BA) sheets are grouped in 5 blocks with each block having 10 or 20 sheets of type $c-g$. In category *block B* a test case has 5 blocks of sheets. Each block consists of 10 or 20 sheets of type h or i. In the *Homogeneous* (H) category a test case consists of all sheets of the same type. The different type of sheets when processed one after another require setups and have different due dates.

D. Results

The number of setups encountered can be reduced by interleaving (recall that setups take significantly more time compared to print time). 4 out of 5 categories in the test set have different types of sheets potentially leading to setups. The heuristic uses the ranking function with weights $\kappa_P = 0.8$, $\kappa_F = 0.15$, $\kappa_{DT} = 0.05$. The weights have been determined by experimentation. Flexibility ensures that the heuristic generates partial schedules that allow rescheduling if required to minimize makespan. Overall, the heuristic generates schedules that have an average difference of 25% from the lower bound. The performance of the heuristic is relatively best for category RB. Figure 6 shows the comparison of the makespans for 5 (randomly selected) test cases from RB shown on the x-axis. The makespan of schedules generated by the schedulers (Eager = E, our heuristic = HS, lower bound = LB) in logarithmic scale is shown on the y-axis. The Eager scheduler does not interleave the sheets based on their types but based on a fixed delay per sheet. The fixed delay does not necessarily minimize the number of setups. Similarly, the MNEH heuristic generates partial schedules that are not flexible resulting in schedules with larger makespan. The heuristic, when computing productivity, minimizes the number of setups. In 56% of cases our heuristic generates schedules with the same or better makespans as the MIP but in much less time (MIP was running for at most 300 seconds).

Figure 7 compares the makespans for category H consisting of 5 (randomly selected) test cases shown on the x-axis and the makespan shown on the y-axis. Due to the

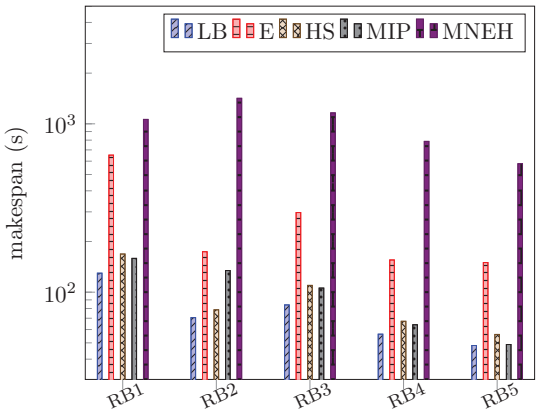


Fig. 6. Comparison of different schedulers for the RB category.

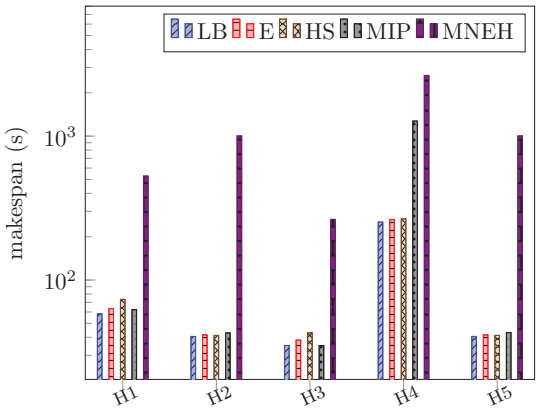
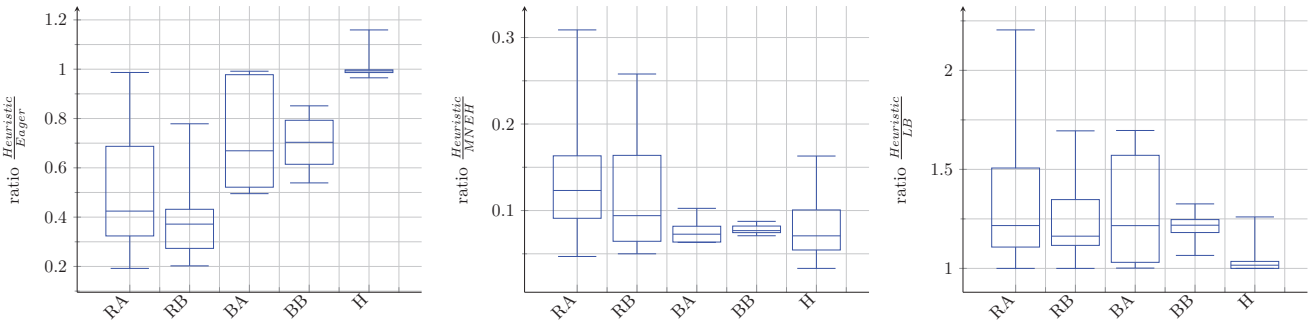


Fig. 7. Comparison of different schedulers for the H category.

similarity of sheets in a job there are no setups required and thus flexible (partial) schedules are not required. Figure 8a compares the heuristic directly to the Eager scheduler. The categories are shown on the x-axis and the y-axis shows the ratio of heuristic makespans to the Eager makespan. For each category, the rectangles show the region where the ratio of makespans of 50% of the test cases in the category fall. The bottom end of the line is the minimum and the top end of the line is the maximum of the ratios observed in the problem set. The line splitting a rectangle is the median of the ratios. A ratio less than 1 indicates that the heuristic performs better. The heuristic out-performs the Eager scheduler in 4 out of 5 categories (i.e. 92% of total test cases). Similarly, Figure 8b compares the makespan ratios of our heuristic with the MNEH heuristic. Our heuristic outperforms the MNEH heuristic in all categories because the MNEH heuristic greedily focuses on maximum productivity and not flexibility. Figure 8c compares the ratios of our heuristic makespans to the lower bounds. The comparison indicates that the heuristic may have room for improvement for the RA category. Recall, however, that the lower bounds are estimates and might not be tight. Our heuristic, to calculate schedules, requires on average 0.3 seconds per sheet and in worst case (for largest schedule request) 6.95 seconds per sheet. On average, over all categories, the proposed heuristic generates schedules that are 40% shorter than the Eager scheduler, have an average difference of 25% compared to the estimated lower bound and generates schedules with less than 67% of the



(a) Compared to the Eager scheduler. (b) Compared to the MNEH heuristic. (c) Compared to the lower bounds.

Fig. 8. Comparison of different schedulers with our heuristic.

makespan of schedules generated by the MNEH heuristic. The calculation time of the proposed heuristic, on average, is 0.3 seconds per sheet where the MNEH heuristic requires 0.01 seconds per sheet to compute schedules trading off runtime over increase in quality of schedules.

VI. Related work

Johnson provides in [4] a polynomial time algorithm to schedule flowshops with two machines and showed that scheduling more than two machines is NP-complete. The CDS heuristic in [5] uses Johnson's polynomial time algorithm to find schedules for flowshops with more than two machines. However, Johnson and the CDS heuristic do not consider re-entrance, setup times and due dates. *Branch and bound* based approaches presented in [6], [7] find optimal schedules for the flowshop scheduling problem. Similarly *mixed integer programs* are used by [8], [9] to solve different variants of flowshop scheduling problem to optimality but are not suitable for online scheduling.

Several heuristics to solve flowshop scheduling problems faster are proposed. For example, by using simulated annealing [10] or genetic algorithms [11]. However, because of due dates in our LSP problem, it is not guaranteed that these algorithms will find a schedule. Other methods to find the schedules faster use a ranking function (as the heuristic presented in this paper). For example, *slope index* based ranking in [12], *rapid access* algorithm in [13] or the NEH heuristic in [1]. These methods rank the jobs to find the near optimal job orderings and do not focus on combination of re-entrance, setup times and due dates. Less attention has been given to re-entrant flowshops. A branch and bound algorithm is used in [14] to schedule a re-entrant flowshop, but it is not suitable for online scheduling. We compared our heuristic with the MNEH heuristic particularly for LSP flow shop instances. Our heuristic outperforms MNEH because our heuristic considers partial schedules that favor rescheduling if better for the later stages of iteration. Evaluation of our heuristic for general re-entrant flowshop as considered in [2] is left as future work.

VII. Conclusion

A re-entrant flowshop scheduling problem is presented with a heuristic to generate online schedules. The heuristic is a greedy strategy which ranks local scheduling decisions and enforces the best ranking choice. The ranking is performed using three metrics, productivity, flexibility and distance. The performance of the heuristic is evaluated by experiments on a test set consisting of schedule requests of industrial relevance. The experiments show that the heuristic outperforms a greedy version of the state-of-the-art MNEH heuristic in all cases. Moreover, the heuristic out-performs the currently used Eager scheduler (for 92% of cases). The run-time of the proposed heuristic is suitable for online scheduling of LSPs.

References

[1] M. Nawaz, E. E. Enscore Jr, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[2] S.-W. Choi and Y.-D. Kim, "Minimizing total tardiness on a two-machine re-entrant flowshop," *EJOR*, vol. 199, no. 2, pp. 375–384, 2009.

[3] H. Emmons and G. Vairaktarakis, *Flow shop scheduling: theoretical results, algorithms, and applications*. Springer, 2012, vol. 182.

[4] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954.

[5] H. G. Campbell, R. A. Dudek, and M. L. Smith, "A heuristic algorithm for the n job, m machine sequencing problem," *Management science*, vol. 16, no. 10, pp. B–630, 1970.

[6] G. McMahon and P. Burton, "Flow-shop scheduling with the branch-and-bound method," *Operations Research*, vol. 15, no. 3, pp. 473–481, 1967.

[7] J. N. Gupta, "A general algorithm for the n× m flowshop scheduling problem," *The International Journal of Production Research*, vol. 7, no. 3, pp. 241–247, 1968.

[8] R. Z. Ríos-Mercado and J. F. Bard, "Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups," *Computers & Operations Research*, vol. 25, no. 5, pp. 351–366, 1998.

[9] E. F. Stafford Jr and F. T. Tseng, "Two models for a family of flowshop sequencing problems," *EJOR*, vol. 142, no. 2, pp. 282–293, 2002.

[10] H. Ishibuchi, S. Misaki, and H. Tanaka, "Modified simulated annealing algorithms for the flow shop sequencing problem," *EJOR*, vol. 81, no. 2, pp. 388–398, 1995.

[11] C. R. Reeves and T. Yamada, "Genetic algorithms, path relinking, and the flowshop sequencing problem," *Evolutionary Computation*, vol. 6, no. 1, pp. 45–60, 1998.

[12] D. Palmer, "Sequencing jobs through a multi-stage process in the minimum total time–a quick method of obtaining a near optimum," *OR*, pp. 101–107, 1965.

[13] D. G. Dannenbring, "An evaluation of flow shop sequencing heuristics," *Management science*, vol. 23, no. 11, pp. 1174–1182, 1977.

[14] D.-L. Yang, W.-H. Kuo, and M.-S. Chern, "Multi-family scheduling in a two-machine reentrant flow shop with setups," *EJOR*, vol. 187, no. 3, pp. 1160–1170, 2008.