Online Scheduling of 2-Re-entrant Flexible Manufacturing Systems

JOOST VAN PINXTEN, UMAR WAQAS, and MARC GEILEN, Eindhoven University of Technology
TWAN BASTEN, Eindhoven University of Technology and TNO-ESI
LOU SOMERS, Océ Technologies and Eindhoven University of Technology

Online scheduling of operations is essential to optimize productivity of flexible manufacturing systems (FMSs) where manufacturing requests arrive on the fly. An FMS processes products according to a particular flow through processing stations. This work focusses on online scheduling of re-entrant FMSs with flows using processing stations where products pass twice and with limited buffering between processing stations. This kind of FMS is modelled as a re-entrant flow shop with due dates and sequence-dependent set-up times. Such flow shops can benefit from minimization of the time penalties incurred from set-up times. On top of an existing greedy scheduling heuristic we apply a meta-heuristic that simultaneously explores several alternatives considering trade-offs between the used metrics by the scheduling heuristic. We identify invariants to efficiently remove many infeasible scheduling options so that the running time of online implementations is improved. The resulting algorithm is much faster than the state of the art and produces schedules with on average 4.6% shorter makespan.

CCS Concepts: • **Theory of computation** → **Scheduling algorithms**; • **Applied computing** → **Multi-criterion optimization and decision-making**; • **Mathematics of computing** → *Combinatorial algorithms*; • **Computer systems organization** → *Embedded and cyber-physical systems*;

Additional Key Words and Phrases: Re-entrant flow shops, flexible manufacturing systems, bounded horizon scheduling

ACM Reference format:
Joost van Pinxten, Umar Waqas, Marc Geilen, Twan Basten, and Lou Somers. 2017. Online Scheduling of 2-Re-entrant Flexible Manufacturing Systems. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 160 (September 2017), 20 pages.
https://doi.org/10.1145/3126551

1 INTRODUCTION

Many flexible manufacturing systems (FMSs) require online scheduling as manufacturing requests arrive on the fly [5, 7, 20]: the timing requirements of a particular operation are only known seconds before the scheduling decision needs to be executed. The scheduling algorithm needs to

This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2017 and appears as part of the ESWEEK-TECS special issue
Authors' addresses: J. V. Pinxten, U. Waqas, M. Geilen, Eindhoven University of Technology, Department of Electrical Engineering, Eindhoven, Netherlands; emails: {j.h.h.v.pinxten, u.waqas, m.c.w.geilen}@tue.nl; T. Basten, Eindhoven University of Technology, Department of Electrical Engineering, Eindhoven, Netherlands and TNO-ESI, Embedded Systems Innovations, Eindhoven, Netherlands; email: a.a.basten@tue.nl; L. Somers, TU/e Technologies, Research and Development, Eindhoven, Netherlands and Eindhoven University of Technology, Department of Mathematics and Computer Science, Eindhoven, Netherlands; email: lou.somers@oce.com.

https://doi.org/10.1145/3126551

compute such decisions in an online fashion. An online scheduler must instruct the FMS in time, such that it does not become a bottleneck for productivity. It is therefore essential to find the *best possible* schedule *in time*.

In this paper we focus on online scheduling for FMSs with *operational/machine flexibility* [3, 17], i.e., the flexibility to create different products with the same machine. Additionally the FMS needs to process products multiple times on the same processing station, where the time between two passes of the same product is considerably higher than the processing time. The flow of the products is fixed, the products need to be produced at the output in a given order, and reordering is not possible. A particular instance of such an FMS is a large-scale printer (LSP) [20]. An LSP prints different types of duplex sheets that need to be processed twice by the same print head at a speed of 100 or more pages per minute. The instructions for how the sheet should flow through the printer therefore need to be provided well within a second. Another example is a chemical process (such as lacquer production [1]) where a certain minimum and maximum time is needed between two subsequent processing steps on the same material. The FMSs under consideration have significant, but limited buffers for re-entrant products. In the LSP the re-entrancy buffer is the return loop within a printer that prepares sheets for printing the opposite side of the sheet. Each sheet has a minimum and maximum travel time determined by physical constraints on heating, cooling, length, acceleration and velocity. Reconfiguration times are significant and have a strong impact on system productivity.

The scheduling freedom in the considered FMSs consists of deciding the order in which new and returning products are processed on the same machine. A schedule needs to meet timing requirements that consist of the travel times between processing stations, and reconfiguration times of the station between processing different products. The most efficient timing can be efficiently derived from the timing requirements [20] using the Bellman-Ford-Moore algorithm [2, 8] once a processing order has been chosen.

The goal of the scheduling algorithm is to find a feasible and maximally productive schedule; the last operation should be executed as soon as possible. Moreover, this schedule should be found within the time budget available for scheduling. However, assessing the impact of sequencing options is not trivial. Interaction between different kinds of products flowing through the FMS makes this a complex problem to tackle.

To address the scheduling problem, we model the FMS as a re-entrant flow shop (see Section 3.1). This paper improves upon the greedy scheduling mechanism of the Heuristic Constraint-based Scheduler (HCS) [20] for online performance by leveraging scheduling properties of the re-entrancy buffer and by applying a generic multi-dimensional meta-heuristic. We introduce the Bounded HCS (BHCS) which improves the greedy scheduling mechanism of HCS by deriving a scheduling horizon from limitations imposed by the re-entrancy buffer to (a) efficiently remove many infeasible sequencing options, (b) calculate timing information only within the scheduling horizon. This leads to much faster evaluation of sequencing options. This speed up allows us to apply the Compositional Pareto-algebraic Heuristic (CPH) [18, 19], a meta-heuristic, to create the Multi-dimensional BHCS (MD-BHCS). MD-BHCS simultaneously explores multiple sequencing options, without resorting to backtracking. This leads to higher quality schedules.

MD-BHCS achieves a better worst-case time complexity, a better average running time and schedules with shorter makespans, on average, compared to the original HCS algorithm. The algorithms, use cases, and benchmark presented in this paper focus on 2-re-entrant flow shops with bounded travel times and reconfiguration times that are significantly larger than the processing times. The solutions can be generalized to flow shops with multiple re-entrances. This needs tuning of some of the algorithmic aspects like the ranking of sequencing options used in the greedy heuristic. This remains future work.

The next section discusses related work. Section 3 precisely defines the problem we address. Section 4 presents the BHCS heuristic. Section 5 combines it with the CPH meta-heuristic. Section 6 contains the experimental evaluation. Section 7 concludes.

2 RELATED WORK

Manufacturing-line scheduling has received a lot of attention in the operations research field. Traditionally, FMSs are modelled as job shops or flow shops. Many specialized exact algorithms and heuristics have been developed to deal with offline and online optimization of job shop and flow shop variants. A plethora of optimization objectives have been defined [10], among which are minimizing makespan [1, 6, 13, 15], total tardiness [11], maximum lateness [7] mean flow time [14], and total completion time [5, 12]. Some approaches can deal only with processing times, others include set-up times, or due dates. Other approaches deal with claiming (multiple) shared resources and simultaneously solve a mapping problem and a sequencing problem [1]. Use cases such as production lines for mirrors, lacquer [1] and textile [13] have been researched. For this paper we assume that the mapping of operations to machines, the flow of operations, and the sequence of jobs is fixed, and that a product is processed twice by one of the machines. The scheduling freedom in our FMS consists of choosing the sequence of operations on the re-entrant machine and calculating efficient schedules.

In our use case, the due dates are relative to the begin times of the operations, and are not allowed to be missed as this will lead to malfunctioning. Relative due dates are more generic than absolute due dates. In most of the related work, due dates are absolute and do not change when the sequence of operations changes. When due dates are involved, most algorithms try to minimize the maximum lateness [7, 9]; i.e the maximum time that a job is completed later than its due date.

Most scheduling problems also allow for arbitrary ordering of the jobs, but in our case it is required to complete them in a specific order. The HCS [20] heuristic addresses the same problem as in this work but it explores a single option, and is therefore more sensitive to the tuning of its internal ranking mechanism. We improve on the current state of the art, HCS. The effectivity of the MD-BHCS algorithm arises from the combination of leveraging necessary properties of feasible schedules for the flow-shop problem with exploring multiple options simultaneously.

Heuristics for job and flow shops such as the Rolling Horizon Procedure (RHP) [5] and decomposition methods [7] have been used to improve the efficiency of exact algorithms by applying them to smaller sub-problems and then implementing a portion of the solution to the sub-problem. This yields close-to-optimal solutions for problems with only sequence-dependent set-up times. Solving a sub-problem with relative due dates with CPLEX may yield optimal solutions for RHP, but does not typically terminate within less than 10 seconds, making it ineffective for online scheduling.

Many common online algorithms such as priority scheduling and rate-monotonic scheduling cannot be applied as we cannot assign a priority to the operations beforehand, and there is no guaranteed cyclic behaviour in the input sequences. They moreover do not take into account the relative deadlines of the operations. Online algorithms such as Earliest-Deadline-First (EDF) and Least-Slack-Time (LST) will schedule the operations in a job-first operation-first order, effectively leading to a single product flowing through the re-entrant buffer. This is ineffective, as in the time between a product's first and second pass the FMS could have used the re-entrant machine to finish other operations. It is also not effective to greedily fill the buffer such that as many products as possible are in the loop, as the sequence-dependent set-up times of a future product possibly cannot be accommodated. This leads to emptying the re-entrant buffer too often. The Nearest Edge Heuristic [11] modified to include relative due dates [20] has also been shown to yield much lower-quality schedules than the current state of the art, HCS.

3 PROBLEM DEFINITION

We first define re-entrant flow shops and introduce terminology and notation. Then we relate (optimal) scheduling of the FMS to the re-entrant flow-shop optimization problem.

3.1 Re-entrant Flow Shops

An FMS is modelled as a re-entrant flow shop with sequence-dependent set-up times and relative due dates.

Definition 3.1 (re-entrant flow shop). A *re-entrant flow shop with sequence-dependent set-up times and relative due dates*, adapted from [20], is a tuple $(M, J, r, O, \phi, P, S, SS, D)$.

- (machines) M is the set of machines that execute the operations.
- (jobs) J is a sequence of jobs. Every job j in J is a sequence of r operations $\langle o_{j,1}, \ldots, o_{j,r} \rangle$ that need to be executed.
- (operations) The operations O of the flow shop are the union of the operations of its jobs: $O = \{o_{j,k} | j \in J, k \in \{1, \ldots, r\}\}$. We write $o_{j,k}$ to denote the k-th operation of the j-th job.
- (re-entrance vector) The re-entrance vector ϕ is a sequence $\langle \mu_1, \ldots, \mu_r \rangle$ with $\mu_i \in M$ of r machines. $\phi(i)$ denotes on which machine the i-th operation of each job is scheduled. The first and second time an operation for a job is executed on machine μ are respectively called the first and second pass of that job j on machine μ, and so forth. This is denoted respectively by $o_{j,\mu,1}$, $o_{j,\mu,2}$, and $o_{j,\mu,i}$. We say that $o_{j,\mu,x}$ is a higher-pass or lower-pass operation than $o_{k,\mu,y}$ when $x > y$ or $x < y$ respectively.
- (processing times) The processing times of operations are denoted by the function $P : O \to \mathbb{R}_{>0}$.
- (set-up times) The set-up times that occur regardless of the sequence of operations on a machine are denoted by a partial function $S : O \times O \mapsto \mathbb{R}_{\geq 0}$, where $S(o_x, o_y)$ is the minimal time needed between the completion of o_x and the beginning of o_y. If $S(o_x, o_y)$ is undefined, no such restriction between the completion of o_x and the beginning of o_y exists.
- (sequence-dependent set-up times) The sequence-dependent set-up times occur only when two operations are executed one immediately after the other on the same machine and are denoted by the function $SS : O \times O \to \mathbb{R}_{\geq 0}$, where $SS(o_x, o_y)$ is the minimal time needed from completion of o_x to beginning of o_y.
- (relative due dates) The due dates impose a maximum time delay between the begin times of two operations, and are denoted by a partial function $D : O \times O \mapsto \mathbb{R}_{>0}$. If $D(o_x, o_y)$ is undefined, then there is no restriction on the maximum time from the beginning of o_x to the beginning of o_y. The algorithms in this paper assume due dates only occur within jobs, i.e., from $o_{x,i}$ to $o_{x,j}$ with $i < j$.

Definition 3.2. A *schedule* $B : O \to \mathbb{R}_{\geq 0}$ for a flow shop describes begin times for all operations; $C(o) = B(o) + P(o)$ denotes the time that operations complete. The following constraints need to be satisfied:

- Every machine $\mu \in M$ can execute at most one operation at a time. That is, $\forall o_x, o_y \in O : (o_x = o_y) \vee (\phi(o_x) = \phi(o_y) \Rightarrow (C(o_x) \leq B(o_y) \vee C(o_y) \leq B(o_x)))$.
- The sequence-independent set-up times of S between each pair of operations o_x and o_y in the domain of S must be met: $B(o_y) \geq C(o_x) + S(o_x, o_y)$.
- If no other operation begins between the begin time of o_x and o_y on the same machine, then the sequence-dependent set-up time between them must hold: $B(o_y) \geq C(o_x) + SS(o_x, o_y)$.

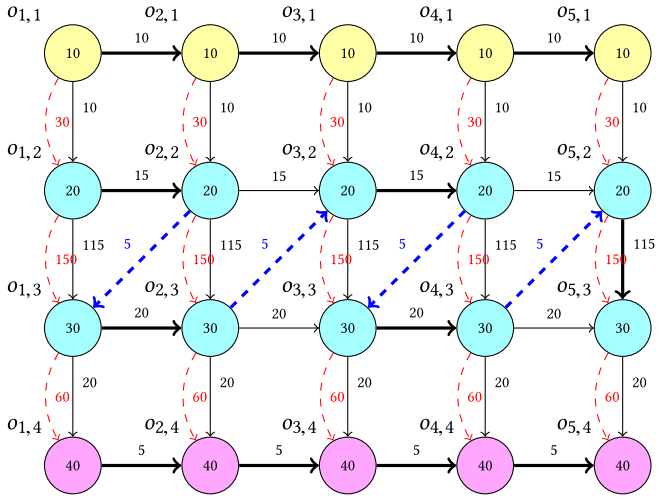

Fig. 1. Example re-entrant flow shop with ordering sequences; the operations are represented by circles, with their processing times inside. Operations with the same colour are mapped to the same machine. Set-up times are shown by black edges, due dates by dashed red edges. Additional set-up times due to the scheduled sequence of operations are shown in dashed blue lines. The effective order of operations on the machines is indicated with thicker lines.

- If a due date $D(o_x, o_y)$ from o_x to o_y is defined, then it imposes a due date on o_y such that $B(o_y) \leq B(o_x) + D(o_x, o_y)$.

A schedule is feasible when all of these constraints are satisfied, and infeasible otherwise.

Definition 3.3 (makespan). The makespan of a schedule B is the latest completion time of any operation:

$$makespan(B) = \max_{o \in O} B(o) + P(o) \tag{1}$$

Definition 3.4 (re-entrant flow-shop optimization problem). An optimal solution to a given flow-shop problem is a feasible schedule B with the smallest possible makespan.

In the algorithms presented in this paper, we consider a single re-entrant machine μ, and job order and operation order are respected. This means that the operation following in the flow vector ϕ cannot begin before all its previous job's operations up to that operation have been executed. For example, it is not allowed that $o_{2,2}$ is executed before $o_{2,1}$ or $o_{1,2}$ (Figure 1). Different passes of different jobs such as $o_{2,2}$ and $o_{1,3}$ do not have a particular order enforced.

3.2 FMS as Re-entrant Flow Shops

The re-entrant flow shops we study are derived from FMSs. The resulting flow shops may have multiple machines, and one machine that processes a product multiple times. Each processing station of the FMS is transformed into a machine, and operations on the products in the processing stations are encoded as operations of the flow shop. The minimum time between two processing steps, due to, e.g., travelling time, reconfiguration or cleaning, is captured by a (sequence-dependent) set-up time between operations. The type of FMSs we focus on in this paper have bounded-time buffering capabilities for a re-entrant machine, which manifest as relative due dates on operations of the same job.

For example, an LSP can be modelled as a 3-machine flow shop with four operations per job, re-entrance vector $\langle \mu_1, \mu_2, \mu_2, \mu_3 \rangle$, and one job for each sheet that needs to be printed [20]. An example with five jobs is shown in Figure 1. Machine μ_1 (first row, yellow) loads a sheet, μ_2 (second row, cyan) prints one side of a sheet, after which it is turned and conditioned in the re-entrancy buffer, before entering μ_2 again (third row, cyan) to print the other side. It is unloaded by machine μ_3 (fourth row, magenta). The load and unload operations must be executed in order of the print sequence. The operations on machine μ_2 need to be ordered such that the system is maximally productive.

In Figure 1 operations are represented by circles, with processing time inside. Between operations of the same job, set-up times enforce the operation order (shown with black, vertical lines). Between the same operation of subsequent jobs, set-up times enforce the job order (shown with black, horizontal lines). The sequence-dependent set-up times (shown with blue dashed lines) enforce a sequence of operations on machine μ_2.

Scheduling a re-entrant flow shop boils down to ordering the operations per machine followed by determining the operation begin times. Figure 1 shows orders per machine in thick lines. The ordering sequences for machines μ_1 and μ_3 are simply the order in which the jobs need to be processed. For machine μ_2 a scheduler needs to obey additional dependencies between re-entrant operations of the machine to ensure that only one operation can be using the machine at any given point in time. The selected order in Figure 1 is $o_{1,2}$, $o_{2,2}$, $o_{1,3}$, $o_{2,3}$, $\ldots$ $o_{5,3}$ and is denoted by thick black and dashed blue lines. The chosen sequence carries additional sequence-dependent set-up times, indicated by the annotations of the blue dashed lines. When the ordering has been determined we can efficiently compute the earliest begin times of the operations by computing a longest path between the first operation and the other operations (see Section 4.2 and 4.4).

4 SCHEDULING APPROACH

We present the outline and describe components of the scheduling approach before investigating them in detail.

4.1 Scheduling Outline

The scheduling outline of HCS [20] is the basis for the BHCS algorithm. BHCS is defined by Algorithms 1 to 6 below. BHCS follows the structure of HCS but deviates most notably in the implementation of Algorithms 4 and 5. (B)HCS is a list scheduling approach that adds one operation into a sequence per iteration until a complete feasible schedule is obtained. It has two stages per iteration; (1) finding (potentially) productive sequencing options for operations on the re-entrant machine, followed by (2) finding the earliest begin times that satisfy all requirements for each of the operations. It then selects one of the feasible options. Finding a maximally productive sequence is a difficult combinatorial optimization problem, but finding the earliest possible begin times once the sequence has been determined is rather efficient (see Section 4.2).

A sequence of operations as constructed by (B)HCS is translated into additional sequence-dependent set-up times imposed on the operations. (B)HCS starts with creating the initial sequence (Algorithm 2) of the first-pass operations and the higher-pass operations of the last job for each re-entrant machine. For example, Figure 2 shows a flow shop f for which CREATE_INITIAL_SEQUENCE(f, μ) generates the sequence $o_{1,1}$, $o_{2,1}$, $o_{3,1}$, $o_{4,1}$, $o_{5,1}$, $o_{5,2}$ for the only machine in the example, μ. These operations must follow a pre-defined order defined by the scheduling constraints. In Algorithms 1, 2, and 3 each sequence s has a schedule t associated with it. The schedule t contains the earliest possible begin times of the operations that respect the sequence s. (B)HCS then iteratively inserts higher-pass operations, such as $o_{2,2}$ in Figure 2, into the sequence s, replacing the necessary sequence-dependent set-up times (e.g., add the edges from $o_{3,1}$ to $o_{1,2}$ and $o_{1,2}$ to

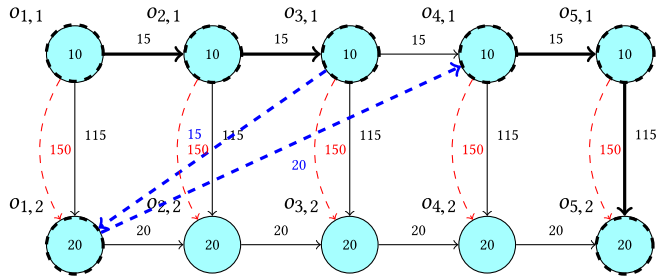

Fig. 2. A sequencing option (dashed arrows), the sequenced operations (dashed outlined vertices) and the corresponding sequence-dependent set-up times (thick edges).

$o_{4,1}$) before checking feasibility of the timing requirements on t. NEXT_ELIGIBLE_OPERATION returns the first re-entrant operation that does not yet occur in the ordering sequence in a job-first operation-first order.

The scheduler extends the sequence of operations by inserting a higher-pass operation of jobs downstream in the input into the current sequence. At each iteration, the partial sequences have updated associated schedules. The schedules contain a lower bound on the begin times of operations. GENERATE_OPTIONS generates options by inserting an operation in multiple places in the given sequence s to produce new partial sequences s' and updates copies of t to a new schedule t'. When all operations of a job have been included in a sequence, the begin times of these operations can be used to instruct the machine to start processing a job. The scheduler finally returns all the begin times associated with the chosen sequence.

ALGORITHM 1: Scheduling Outline of BHCS

```
1:  function SCHEDULE(flow shop f, re-entrant machine μ, ranking RANK)
2:      (s, t) = CREATE_INITIAL_SEQUENCE(f, μ)
3:      repeat
4:          o_e = NEXT_ELIGIBLE_OPERATION(f, s)
5:          // Generate feasible options and update operation times
6:          l = GENERATE_OPTIONS(f, o_e, (s, t))
7:          // Select the best result from the set
8:          (s, t) = SELECT_BEST(RANK, f, l, o_e)
9:      until all re-entrant operations of μ included in s
10:     return t
```

The RANK function computes an absolute assessment for each partial solution based on a number of metrics from which (B)HCS greedily selects the option with the best assessment. The metrics used in our scheduler are introduced and explained in Section 5.2.

For each eligible operation, the sequencing options are created and evaluated in Algorithm 3. It creates a list l of feasible sequences and associated schedules by inserting the eligible operation before any of the operations returned by Algorithm 4, evaluating them, and adding them to the list only when they are feasible.

The begin times are calculated using the Bellman-Ford-Moore (BFM) [2, 8] longest-path algorithm on a directed graph (Section 4.2) created from the combination of the processing times, set-up times, and the relative due dates. When BFM detects positive cycles in this graph, then the sequence has no feasible schedule.

ALGORITHM 2: Create Initial Sequence and Initialize Begin Times

```
1: function CREATE_INITIAL_SEQUENCE(flow shop f, re-entrant machine μ)
2:     s = ⟨⟩
3:     // Add first passes of all jobs
4:     for each i = 1 to |J| do append o_{i,μ,1} to s
5:     // Followed by re-entrant operations of last job
6:     for each i = 2 to |r| do append o_{|J|,μ,i} to s
7:     // Initialize all begin times associated with s
8:     for each o ∈ O(f) do t[o] = 0
9:     return (s, t)
```

ALGORITHM 3: Find and Apply All Sequencing Alternatives, and Update Begin Times Accordingly

```
1: function GENERATE_OPTIONS(flow shop f, operation o_e, ordering sequence with associated
   begin times (s, t))
2:     l = ∅
3:     for each o_p in FIND_INSERTION_POINTS(f, s, o_e) do
4:         s' = copy of s
5:         insert o_e into s', before o_p
6:         (t', v) = UPDATE_BEGIN_TIMES(f, s', t)
7:         // If v is true, the schedule is feasible so far
8:         if v then
9:             l = l ∪ {(s', t')}
10:    // Return the set of feasible sequences and associated times
11:    return l
```

In the following sections, we introduce these ingredients in more detail. In particular, we contribute two invariants that allow us to bound the execution time of the online algorithm. These two invariants allow us to define BHCS, which significantly improves the worst-case time complexity of HCS. We also introduce new metrics that improve the quality of the schedules. The first invariant purges many infeasible sequencing options (Section 4.3). The second invariant allows us to calculate begin times for only a bounded horizon and to still guarantee detection of infeasible sequencing options (Section 4.4). This results in an online adaptation of the Bellman-Ford-Moore longest-path algorithm.

4.2 Computing Operation Begin Times From a Sequence

The decisions made by the scheduling algorithm (Algorithm 1) ensure that all re-entrant operations are eventually scheduled into a single sequence per machine. Such a sequence imposes the required additional sequence-dependent set-up times, shown as dashed blue lines in the example in Figure 2. A schedule of begin times can be computed from the sequence of operations by a longest-path algorithm. The earliest possible begin times for the given sequence are the longest-path time distances of all operations starting from the initial operation [16].

The graph for longest-path computation consists of vertices representing the begin times of operations, and edges that impose constraints. An edge from o_x to o_y with weight δ in the converted graph means that $B(o_y) \geq B(o_x) + \delta$. The constraints of Definition 3.2 are translated from

ALGORITHM 4: Find the Set of Sequencing Options

```
1:  function FIND_INSERTION_POINTS(flow shop f, sequence s, operation o_{i,k})
2:      r = ∅
3:      o_p = o_{i,k−1}
4:      slack = smallest transitive due date in D on o_p
5:      // Add all successors until slack is non-positive
6:      while slack > 0 ∧ (s has operation after o_p) do
7:          o_n = NEXT(o_p, s)
8:          r = r ∪ {o_n}
9:          d = smallest transitive due date in D on o_n
10:         slack = min(slack - P(o_p) − SS(o_p, o_n) , d)
11:         o_p = o_n
12:     return r
```

the processing, (sequence-dependent) set-up times, and relative due dates as follows:

$$B(o_y) \geq B(o_x) + P(o_x) + S(o_x, o_y)$$

$$B(o_y) \geq B(o_x) + P(o_x) + SS(o_x, o_y)$$

$$B(o_y) \leq B(o_x) + D(o_x, o_y) \Leftrightarrow B(o_x) \geq B(o_y) - D(o_x, o_y)$$

The resulting graph looks similar to Figure 1, except that the red dashed edges corresponding to due dates are reversed, and their values are negated, and that the processing times are propagated into all of the vertex's corresponding set-up times [20].

4.3 Purging Infeasible Sequencing Options

We can determine the sequencing options for each re-entrant operation (Algorithm 4) by removing infeasible sequencing options. We use the property that an eligible operation $o_{j,x}$ must obey the operation sequence for job j and also obey the job sequence for the operation x. So $o_{j,x}$ cannot begin *before* any of its predecessors $o_{i,y}$ such that $i \leq j \wedge y \leq x$, nor *after* any of its successors $i \geq j \wedge y \geq x$. Consider an FMS with one re-entrant machine where three passes of a job need to be scheduled on a single machine. In the example of Figure 3 all hatched operations cannot be sequenced immediately before operation $o_{3,2}$ due to job and operation sequence constraints.

Additionally, we use the invariant that the scheduling algorithm does not reorder operations in the scheduling sequence; it will only insert more operations into this sequence. This invariant allows us to determine which scheduling option is the last possible insertion point in the given sequence using static information about set-up times and due dates. This allows us to conclude infeasibility without calculating exact earliest begin times. The minimum time approximation in Algorithm 4 is used to determine which due date is the first to be certainly violated. The algorithm traverses the sequence starting from the source of the due date $o_{i,k-1}$ by iteratively investigating the NEXT operation in the sequence. NEXT returns the immediate successor of o_p in the sequence s. Summing the processing and set-up times for the given sequence provides a lower bound on the time difference between the given operation and its source in any valid schedule.

For example, operation $o_{3,2}$ cannot begin after operation $o_{5,1}$, $o_{5,2}$ or $o_{5,3}$, as this would violate the due date from $o_{3,2}$ to $o_{3,1}$; the lower-bound time differences following the given sequence between $o_{3,1}$ and these operations are respectively 170, 290, and 415, which are all more than the due date constraint of 150. When this lower bound on the minimum time between two operations exceeds the tightest due date encountered, then any schedule containing this ordering will be infeasible.

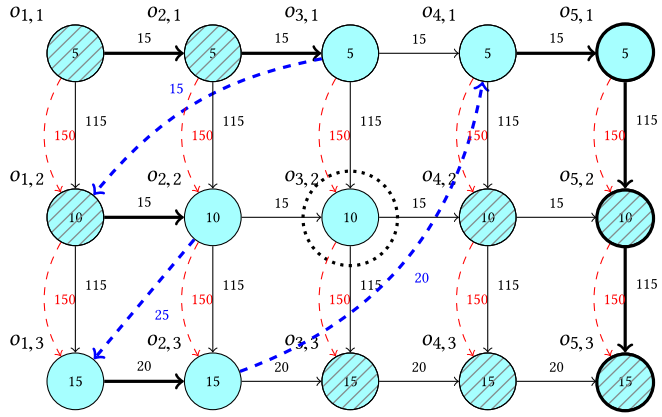

Fig. 3. Inserting operation $o_{3,2}$ into the sequence immediately after hatched operations (marked with diagonal lines) is infeasible due to sequence constraints on jobs and operations. Sequencing $o_{3,2}$ anywhere after $o_{4,1}$ (operations with thick border) is infeasible due to violation of the due date of $o_{3,1}$ to $o_{3,2}$.

Scheduling after the last possible operation is definitely infeasible as we found that the minimum set-up time to do so is already larger than the due date imposed on the scheduled operation. As all the processing and set-up times are non-negative, we cannot insert more lower-pass operations in the re-entrant buffer when the due date would already be violated. The due date with the least slack typically corresponds to the smallest re-entrant buffer time in the FMS. In other words, the number of sequencing options to be considered is bounded by the ratio L which is the largest buffer time of the FMS divided by the smallest processing time.

If we would not take into account this property, we would end up with $O(|J|)$ options per iteration. The bound described in this subsection allows us to stop generating options as they would end up as infeasible schedules. They would be detected by checking for positive cycles in a graph, which incurs the worst-case runtime of the BFM algorithm. It is, therefore, much more efficient to avoid evaluating them. The difference in worst-case complexity with and without purging is studied in more detail in Section 4.5.

4.4 Bounded Horizon

For online scheduling, it is only necessary to find feasible (and hopefully productive) begin times for the operations that are to be executed next. BHCS can defer computing the begin times of operations that do not influence the current decision. Such operations can still be indefinitely postponed and do not influence the feasibility of the current sequencing option.

To detect infeasibility of a sequencing option, BHCS needs to include at least those edges that are potentially involved in a positive cycle. Only the first of the two edges introduced by a sequencing option (such as $o_{3,1}$ to $o_{1,2}$ in Figure 2) creates additional cycles. To check for positive cycles, BFM needs to check the sub-graph induced by all operations in jobs between the job of the sequenced operation and the job of its predecessor in the sequence. All operations of these jobs are included, as they may have intra-job due dates associated with them.

The sources V_s are the operations which have been fully scheduled before. They indicate when the machines become available to process the next operations. They are all operations that have set-up times into that smallest sub-graph, but are not included in that smallest sub-graph. For example, the smallest sub-graph for scheduling and checking feasibility of inserting $o_{2,2}$ between $o_{4,1}$ and $o_{5,1}$ in Figure 2 consists of the operations of jobs 2 to 4. Its sources are the operations of

job 1. The begin times for the sources is not allowed to be changed, as these operations may have been executed already.

Although the longest-path computation is computationally efficient, it is still the most time-consuming element of the scheduling algorithm when the job set becomes larger. Typical speed-up approaches such as GPU-acceleration [4] can improve the worst-case execution time by a factor 50 to 100. However, this gain is not enough as the number of jobs can increase arbitrarily. Such accelerations can be implemented orthogonally, but without bounding the sub-graph to be analysed, the longest-path computation will always become a limiting factor to the online applicability of the algorithm. We use the bounded subset of operations described in the previous paragraph as input for the modified BFM algorithm, defined in Algorithm 5, to reduce the computational effort to evaluate a scheduling option.

ALGORITHM 5: Bellman-Ford-Moore with Multiple Sources

```
1:  function BFM(Graph G(V, E, w), subset of active operations V_a ⊂ V, set of sources V_s ⊂ V,
    begin times for sources d')
2:      for each a ∈ V_s do
3:          d[a] = d'[a]
4:          for each (a, v) ∈ E do
5:              RELAX(d, a, v, w)
6:      f = true // assume feasible
7:      E_a = {(x, y) ∈ E : x ∈ V_a}
8:      for each i ∈ {1, ..., |V_a| − 1} do
9:          for all (u, v) ∈ E_a do
10:             r =RELAX(d, u, v, w)
11:             if v ∈ V_s ∧ r then f = false
12:     for each edge u, v ∈ E_a do
13:         if (RELAX(d, u, v, w) then f = false
14:     return f, d
```

ALGORITHM 6: Relax One Edge

```
1:  function RELAX(d, u, v, w)
2:      if d(u) + w(u, v) > d(v) then
3:          d(v) = d(u) + w(u, v)
4:          return true
5:      return false
```

We assume that the initial graph without sequence-dependent set-up times is feasible, and that we schedule one operation at a time. Inserting an operation in the sequence of scheduled operations introduces additional set-up times. An additional positive cycle must include at least one of the two edges introduced by the scheduling option, the processing and set-up times, and at least one due date. As we consider only intra-job due dates, operations of jobs later than the insertion point cannot create additional positive cycles. The begin times of operations in jobs before the eligible operation's job are not allowed to be changed (line 10 in Algorithm 5), as changing them could lead to an infeasible schedule through a positive cycle that is outside the sub-graph.

When we consider the invariant that at the beginning of Algorithm 5 all times for the scheduled operations were feasible, then it can detect infeasibility within the horizon when the set of sources consists of the operations of the last-scheduled job plus operations containing sequence-dependent set-up times into the active vertices, and the set of active vertices consists of the operations of all jobs that cannot be delayed indefinitely any more. The jobs that can be delayed indefinitely are the ones starting from the insertion point, e.g., operation $o_{4,1}$ in Figure 2. At the end of Algorithm 5 we either find that the sequence is infeasible, or we find feasible begin times, which can be used in the next iteration as initial estimates to speed up convergence of the longest-path computations.

If a sequence is considered feasible, then for an FMS with one 2-re-entrant machine this is enough to guarantee feasibility in the overall problem. The sequence-dependent set-up times that are necessary to empty the loop from higher-pass operations returning from the buffer are already included as constraints, and have already been verified to not violate due dates. The set-up time after emptying the buffer starts an operation which could still be delayed indefinitely and, therefore, such a decision will not violate any due date. When feasible begin times are returned for this subset, it is guaranteed that this partial schedule is feasible for the overall problem with this sequence. For higher-re-entrancy FMSs or multiple re-entrant machines however, the heuristic might select a feasible sequence that does not have any feasible follow-up option.

Algorithm 5 runs in $O(|V_a| \cdot |E_a|)$ where V_a is the set of active vertices, and E_a its corresponding set of edges. This helps us to greatly decrease the complexity of the scheduling algorithm.

4.5 Worst-case Time Complexity of HCS and BHCS

We assess the impact of the improvements by comparing the worst-case time complexity of HCS to BHCS for flow shop instances that are derived from the structure of an FMS. Recall that $|J|$ denotes the number of jobs and r the number of operations per job. The number of vertices in the converted graph (Section 4.2) is $O(|J| \cdot r)$ and the number of edges is also $O(|J| \cdot r)$ as the number of edges per vertex is bounded. We assume there are no inter-job due dates, and set-up times occur only between operations of the same job, between same-pass operations of subsequent jobs and in sequencing options.

The worst-case time complexity of HCS occurs when the last evaluated option is the only feasible option, as it then encounters the worst-case execution time for BFM, which is $O(|V| \cdot |E|) = O(|J|^2 \cdot r^2)$, for each scheduling option. The number of options to be evaluated may grow in the worst case with the size of the partial sequence for each re-entrant operation, $O(\sum_{i=1}^{|J| \cdot r} i) = O(|J|^2 \cdot r^2)$, the worst-case complexity for HCS is $O(|J|^4 \cdot r^4)$.

BHCS creates at most L sequencing options for each of the $O(|J| \cdot r)$ re-entrant operations (see Section 4.3). The largest subset of edges and vertices that needs to be used to detect infeasibility is then of size $O(L \cdot r)$, and we only need to use a limited set of operations as sources, the worst-case complexity of Algorithm 5 is $O(|V_a| \cdot |E_a|) = O(L^2 \cdot r^2)$.

Therefore, the worst-case complexity of BHCS is the number of re-entrant operations to be scheduled ($|J| \cdot r$) multiplied by the number of sequencing options to be evaluated ($|L|$) multiplied by the evaluation cost of one option ($L^2 \cdot r^2$); totalling $O(|J| \cdot L^3 \cdot r^3)$, where $r \ll |J|$ and $L \ll |J|$ in typical cases. The reduction in evaluation of options and the bounded horizon lead to a worst-case complexity of BHCS which is linear in the number of jobs and is therefore preferred over the super-linear complexity of HCS.

5 EXPLORING TRADE-OFFS IN SCHEDULING DECISIONS

We apply the CPH multi-dimensional meta-heuristic [18, 19] to BHCS to increase the quality of generated schedules by increasing the search space. We introduce the MD-BHCS (Section 5.1) and

define new metrics to rank schedules (Section 5.2). In the scheduling outline (Algorithm 1, line 8) we implicitly used the idea of good schedules. These metrics are used in a linear combination in (B)HCS to select a single option greedily. They are used as separate metrics in MD-BHCS to explore multiple Pareto-optimal options simultaneously.

5.1 Scheduling Outline with CPH

CPH allows the scheduler to explore multiple sequencing options simultaneously, while still limiting the computation time. The goal of the CPH meta-heuristic is to consider (all or a representative subset of) the Pareto-optimal sequencing options, in other words, to eliminate those options for which there is some other option that is at least as good in any of the metrics and strictly better in at least one metric. Note that two sequencing options may have the same values for all metrics. If they are Pareto optimal then both options are kept. Where the combination of metrics in (B)HCS is used to select a greedily single (Pareto-)optimal trade-off, MD-BHCS explores multiple Pareo-optimal trade-offs.

ALGORITHM 7: MD-BHCS

```
1:  function SCHEDULE(flow shop f, re-entrant machine μ, size of partial solutions set k)
2:      g = {CREATE_INITIAL_SEQUENCE(f, μ)}
3:      repeat
4:          o_e = NEXT_ELIGIBLE_OPERATION(f, arbitrary s ∈ g)
5:          g' = ∅
6:          for each (s, t) ∈ g do
7:              // Generate feasible options, update operation times
8:              l = GENERATE_OPTIONS(f, (s, t), o_e)
9:              g' = g' ∪ l
10:         g = minimize(g')
11:         reduce size of g to k
12:     until all re-entrant operations of μ included in g
13:     return s ∈ g with the smallest makespan
```

MD-BHCS (Algorithm 7) generates partial solutions as follows. Similar to BHCS, the initial sequence is initialized before any operation is scheduled and it is the starting point for all partial solutions that will be explored. A set of partial sequences is explored, starting with only the initial sequence. The outer loop ensures that each sequence eventually contains all re-entrant operations. The inner for-loop creates a new generation of sequences by scheduling one eligible operation, starting from each of the previous generation's sequences in g. Algorithm 5 is again used to evaluate feasibility of sequencing options and to compute the begin times of operations.

Instead of selecting a single 'best' option, we make a multi-dimensional assessment of the partial solutions. The set of partial solutions is minimized through the Pareto-cull process which removes all dominated (non Pareto-optimal) solutions from the set. This enables us to restrict the search space to only those partial solutions having valuable trade-offs.

When many of the partial solutions are Pareto-optimal then the set of partial solutions may grow very large. This can be prohibitive for online performance. However, as partial solutions with similar metric values are likely to produce similar results, we approximate the full set with a subset of bounded size which is as diverse as possible in the considered metrics. The number of partial solutions to consider is a parameter of the MD-BHCS algorithm. Keeping more partial solutions typically leads to a higher runtime but also to a higher schedule quality.

5.2 Assessing Partial Solutions

We divide a schedule into three parts for the assessment: (1) the part containing jobs for which all operations are sequenced and should be executed as fast as possible, (2) the part that reflects the impact of the sequencing options on the state of the re-entrancy buffer, and (3) the part which can still be indefinitely postponed.

MD-BHCS assesses these three parts with metrics that can be readily obtained from the begin times provided from Algorithm 5 and that together characterize partial solutions. For the last inserted operation o, and a sequence s with an associated partial schedule B:

- *past work* is assessed by measuring the earliest possible begin time B for o: $P(B, o) = B(o)$ (lower is better),
- *committed work* is assessed by the earliest possible begin time for the operation immediately following o in the scheduled sequence: $W(B, o) = B(\textsc{next}(o, s))$ (lower is better),
- *future work* is assessed by the productivity of the remaining part (higher is better). We approximate it by the number of operations $nr\_ops(s, o)$ that can be delayed indefinitely (lower is better).

The metrics influence the makespan of the final schedule as follows. Past work compares the influence of a scheduling option on the begin time of the eligible operation, taking into account how much buffer time is used by processing lower-pass operations. The metric favours sequences for which the last inserted operation begins earlier. Delaying the last inserted operation is only interesting when we would benefit in the future from avoiding work or penalties for unsequenced operations. The committed work indicates the amount of work added to the re-entrant buffer, trying to minimize the work that needs to be done directly after this option. Future work measures how many units of work remain after all the committed work has been done, favouring less work. These metrics are heuristics to determine the different qualities of a sequencing option for the final makespan. We evaluate the metrics through the schedule quality in Section 6.3 by comparing the different variants of HCS to lower bounds.

5.3 Reducing the Set of Candidate Solutions

We reduce the set of partial solutions such that it contains at most a pre-defined maximum number of partial solutions k. We have used the archive truncation method from the Strength-Pareto Evolutionary Algorithm (SPEA2) [21], which iteratively removes the solution which has the smallest distance to other solutions, until there are only k partial solutions left. To reduce the impact of different magnitudes of the metrics on the distance, we normalize each metric between 0 and 1 by scaling them linearly to fit between their minimum and maximum observed value.

The normalized metrics are used in the ranking function of BHCS which defines the respective relevance of the metrics. In the Pareto-cull process of MD-BHCS the metrics are used without normalizing them, as the relative weight has no effect on Pareto optimality.

5.4 Worst-case Time Complexity of MD-BHCS

The main difference between BHCS and MD-BHCS is in the number of the partial solutions and how they select their next generation of sequences. Whereas BHCS starts each iteration with one solution and generates at most L solutions to evaluate, MD-BHCS starts with k solutions and generates at most $k \cdot L$ solutions for each of the $O(|J| \cdot r)$ re-entrant operations. The Pareto minimization uses the Pareto-cull operation and takes at most $O(k^2 \cdot L^2)$. The reduction process of SPEA2's archive truncation method for a solution set of size M has a worst-case time complexity of $O(M^3)$ [21], where $M \leq k \cdot L$. The minimization, reduction, and evaluation of begin times are

all in the inner loop and are executed for each iteration. The complexity of evaluating the begin times of all $k \cdot L$ options takes $O(k \cdot L \cdot L^2 \cdot r^2)$. The worst-case time complexity of MD-BHCS is therefore $O(|J| \cdot r \cdot k \cdot L^3(k^2 + r^2))$.

As k, r and L are constants, the algorithm runs in time linear to the number of operations $|J|$ in the flow-shop problem. The worst-case time complexity of BHCS as explained in Section 4.5 is $O(|J| \cdot L^3 \cdot r^3)$. The additional complexity of MD-BHCS over BHCS for evaluating more options is at most cubic in k.

6 EXPERIMENTAL EVALUATION

In this section we describe the experimental set-up, compare the makespan of the generated schedules, and evaluate the runtime of the schedulers.

6.1 Experimental Set-up

We have implemented the two scheduling-algorithm variants BHCS and MD-BHCS. As in the original HCS implementation, the processing times, set-up times and due dates are encoded as fixed precision values to avoid rounding errors that are typical for floating point implementations. All experiments have been executed on the same 8-core Intel i7 running at 3.0 GHz. All algorithms are implemented as single-thread programs.

The 3-machine, 2-re-entrant benchmarks representing print requests for the LSP as introduced in [20] are used to assess the quality of the schedules and the runtime of the algorithms. The benchmark consists of 85552 sheets in 701 print requests. Each print request is taken from one of the following categories:

- H Homogeneous: repetition of one sheet type
- RA Repeating A: repetition of one sheet type followed by another sheet type
- RB Repeating B: repetition of one to three sheets of a type followed by another sheet type
- BA Block A: 5 blocks of 5 different sheet types, each block contains 10 or 20 sheets of the same type
- BB Block B: 5 blocks with two alternating sheet types, each block contains 5 to 25 sheets of the same type

We compare the makespans with the makespans of schedules generated by HCS and the results of a mixed integer programming (MIP) formulation of the scheduling problem. While searching for the optimal solution for the MIP, CPLEX also computes the optimal objective value for linear relaxations of the MIP. Such relaxations give a lower bound for the makespan of the MIP. A schedule computed for a linear relaxation typically violates integer feasibility constraints, and as such does not necessarily represent a solution to the original problem. When our heuristic or CPLEX finds a schedule that has a makespan equal to the lower bound, then the bound is exact and that schedule is proven to be optimal. Otherwise, it is unknown how close the lower bound is to the actual optimum. The lower bounds for the benchmarks have been calculated with CPLEX 12.6.0 with a time limit of 1000 seconds (pre-solve and solve).

We used a weighted combination of the normalized metrics for the ranking mechanism of BHCS. The weights were chosen empirically by selecting initial weights and tuned by iteratively applying small variations and re-scheduling the full benchmark. Tuning was repeated until no further improvements in makespan were found. The best results by BHCS, for our benchmark, were found using the following weighted combination for a given schedule B, associated sequence s and last inserted operation o:

$$rank(B, o, s) = 0.3 \cdot \textsc{norm}(P(B, o)) + 0.6 \cdot \textsc{norm}(W(B, o)) + 0.1 \cdot \textsc{norm}(nr\_ops(s, o)) \quad (2)$$

NORM ensures that a dimension is normalized between 0 (best observed) and 1 (worst observed).

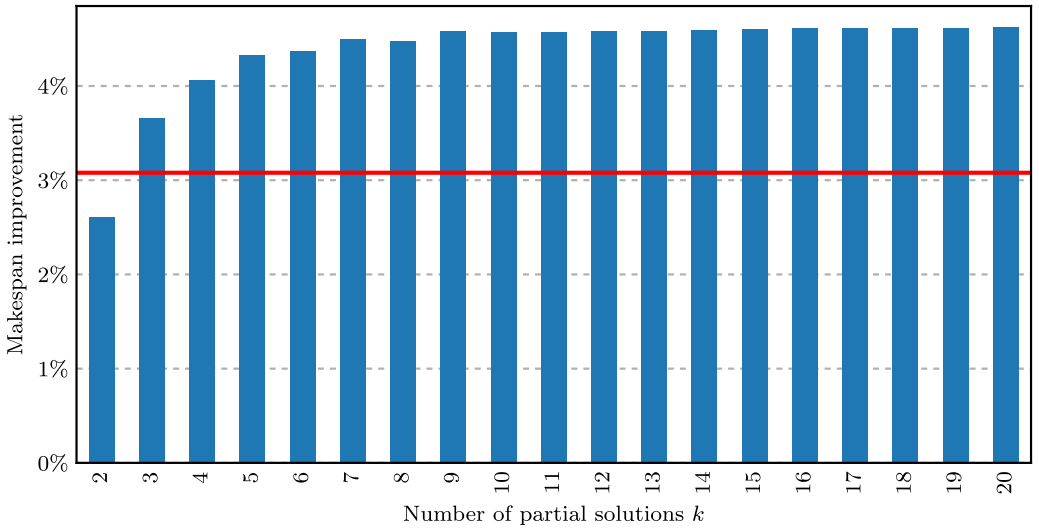

Fig. 4. Average makespan improvement of MD-BHCS (vertical blue bars) and BHCS (horizontal red line) over HCS.

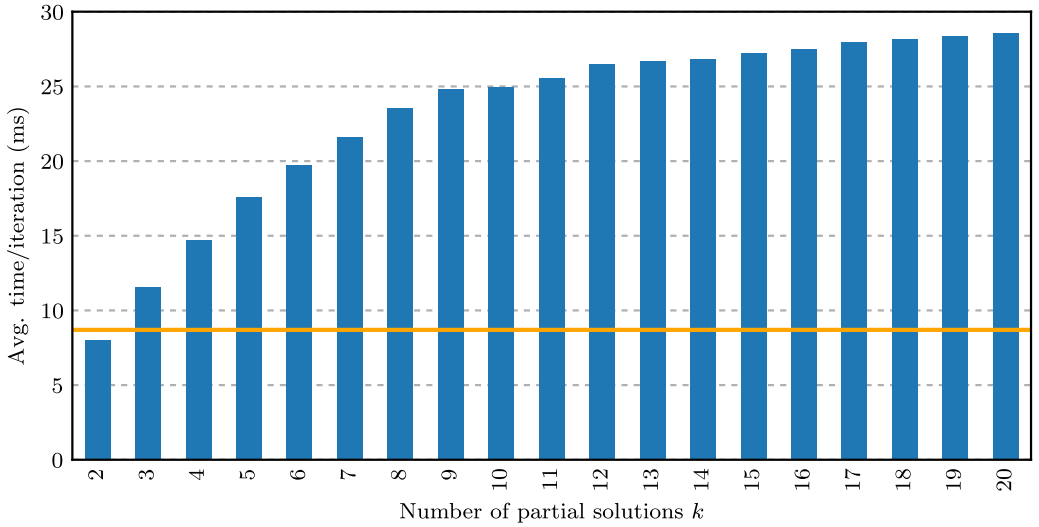

Fig. 5. Average time per iteration for MD-BHCS for varying k. The horizontal orange line is the average time per iteration for BHCS.

6.2 Sensitivity to Number of Partial Solutions

The performance of the MD-BHCS algorithm, both in quality (Figure 4) and runtime (Figure 5), depends on the number of partial solutions k. The makespan of a schedule typically becomes shorter when parameter k becomes higher, while the average time per iteration increases. MD-BHCS with $k = 2$ is faster than BHCS and also produces worse schedules. With $k = 2$, the scheduler can only account for two extremes in a three-dimensional assessment and, therefore, cannot cover the trade-offs accurately. Additionally, BHCS' weighted sum does not focus on the extremes of the

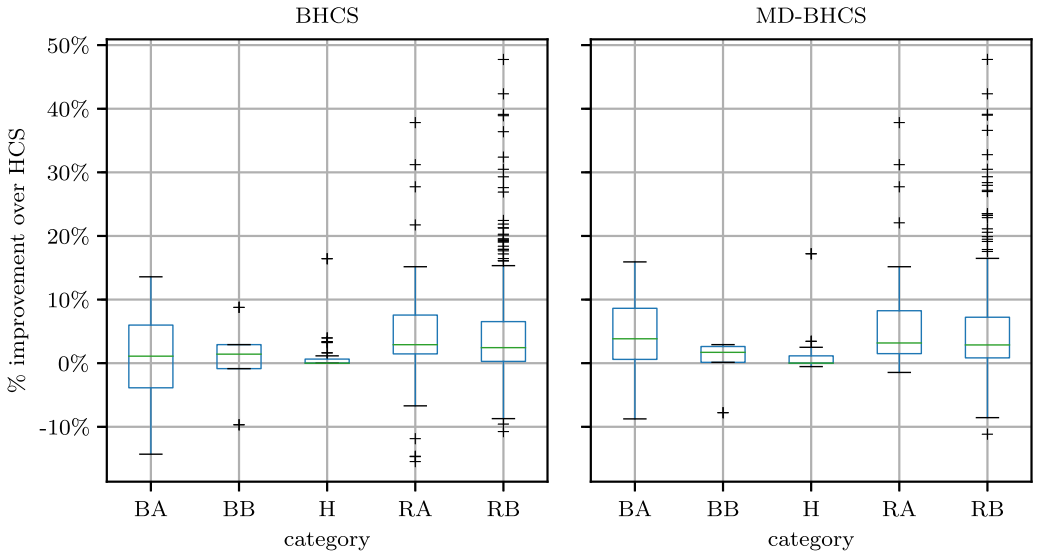

Fig. 6. Makespan improvement of BHCS and MD-BHCS ($k = 20$ solutions) relative to HCS (higher is better).

trade-offs. They are likely to be too aggressive in one aspect, for example, aggressively filling the buffer. The generated Pareto points for $k = 2$ are less likely to have many follow-up options, and typically lead to fewer evaluations of infeasible sequences.

The quality benefits of CPH start to diminish when $k = 10$ or more partial solutions; the schedules improve only slightly, but at the cost of additional runtime. Figure 4 shows that when $k = 20$ partial solutions are taken into account, MD-BHCS produces the best makespans. The effect of exploring multiple options simultaneously with the meta-heuristic does not increase the observed runtime significantly, as the number of Pareto-optimal partial solutions is often less than k.

6.3 Makespan Comparison

The box plot in Figure 6 shows the improvement over the makespans from HCS. The median improvement lies between 0% and 4%, with several outliers over 20%. The difference in the H and BB category is very small as the set-up times in all of these job sets are very small. Both schedulers manage to keep the buffer occupied for these cases which produces close-to-optimal results. For the other categories, MD-BHCS typically out-performs HCS.

Table 1 shows how close the results of different variants of HCS are to the CPLEX lower bounds. The first two columns show the average percentage above the lower bound; column $CPLEX_{all}$ includes all benchmark instances, and column $CPLEX_{opt}$ includes only those instances for which CPLEX found an optimal solution. In the *OPT* column, the number of solutions which have makespan equal to their lower bound are shown. These results show that the makespans are over 15% higher than the CPLEX lower bounds ($CPLEX_{all}$). However, when we compare only the cases where CPLEX found optimal schedules, ($CPLEX_{opt}$), we see that the results are within 1% of the optimal result. CPLEX found 108 optimal schedules, and the different schedulers found several optimal schedules. The HCS variants sometimes manage to find optimal schedules when CPLEX only determined a lower bound but could not determine whether it is indeed possible to achieve such a schedule.

Table 1. Makespans Compared to CPLEX Lower Bounds; $CPLEX_{all}$ Is Average Percentage Above the Lower Bound (All 701 Cases), $CPLEX_{opt}$ Is Average Percentage Above Lower Bound for 108 CPLEX Optimal Cases, OPT is Number of Cases Where the Makespan is Known to Be Optimal

		$CPLEX_{all}$	$CPLEX_{opt}$	OPT
HCS		21.90%	4.40%	39
BHCS		18.26%	1.48%	48
MD-BHCS	k=2	18.96%	2.33%	40
	k=6	16.87%	0.64%	74
	k=10	16.60%	0.62%	80
	k=20	16.54%	0.62%	80

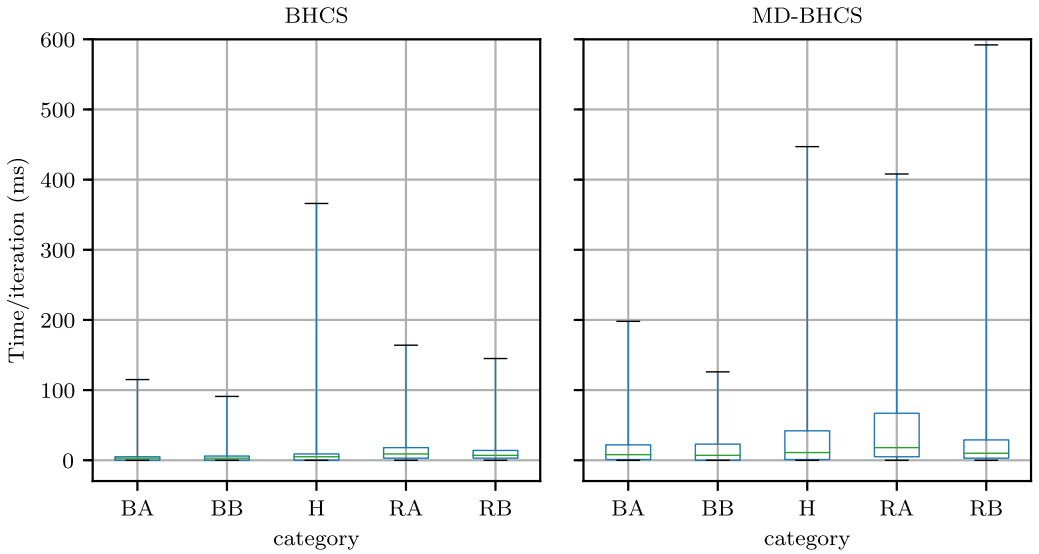

Fig. 7. Runtime per scheduling decision for BHCS and MD-BHCS (outliers included in whiskers).

6.4 Runtime Evaluation

The runtime per category in Figure 7 for this benchmark with $k = 20$ is below 600 ms, with an average runtime of 28 ms. This shows that the proposed algorithm is indeed fast enough for online computation of schedules for the LSP, indicating that it can be applied to re-entrant flow shops originating from FMSs. Figure 7 also shows that the observed worst-case running time is a factor 10 higher than the median, due to the infeasibility checks still incurring worst-case behaviour. Due to the large data set the many outliers have been included in the whiskers.

We have also compared the runtime of our implementations to the implementation of HCS [20] and noticed that the difference in runtime can become arbitrarily large. In the benchmark, the runtime is up to 100 times slower than BHCS. The observed runtime of HCS scales super-linearly with the number of jobs in the input. The average processing time per scheduling decision in the complete benchmark is 240ms for HCS and 8.6ms for BHCS, on average 28 times faster. For only 14% of the cases, HCS is up to twice faster than BHCS. MD-BHCS with $k = 20$ is 3 times slower than BHCS and, therefore, only 9 times faster than HCS on the whole benchmark.

7 CONCLUSION AND FUTURE WORK

We have shown that we can drastically reduce the worst-case time complexity and average running time of scheduling re-entrant flow shops that originate from FMSs by investigating a smaller part of the timing of operations, and avoiding the evaluation of infeasible sequencing options. We have used properties of the scheduling problem to quickly remove these infeasible options by inspecting local information. We have also shown that the worst-case time complexity depends on the maximum number of products that fit simultaneously in the buffer of an FMS.

The performance of the multi-dimensional meta-heuristic CPH on this flow shop scheduling problem confirms it is a promising meta-heuristic for online scheduling. MD-BHCS produces on average 4.6% shorter makespans than the previous state of the art, and is 9 times faster. Trading off solution quality for runtime is useful to avoid the scheduler becoming a bottleneck for productivity.

For future improvements, it would be interesting to incorporate solution diversity in the reduction operator. The current implementation selects representatives based on the assessment of partial solutions, but does not consider diversity or distance in the solution domain. It would also be interesting to make this algorithm an any-time algorithm, such that it provides scheduling solutions before completing the calculation of the new generation. One step in this direction would be to remove parameter k and evaluate options according to the time budget given. This could be achieved by reordering the evaluation of sequencing options, such that the most promising trade-offs are evaluated before the time budget runs out.

ACKNOWLEDGMENTS

This work is part of the research programme Robust CPS with project number 12693 which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). We thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] Gerd Behrmann, Ed Brinksma, Martijn Hendriks, and Angelika Mader. 2005. Production scheduling by reachability analysis: a case study. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International.* IEEE, Los Alamitos, CA, USA, 140–142. DOI: http://dx.doi.org/10.1109/IPDPS.2005.363

[2] Richard Bellman. 1958. On a routing problem. *Quarterly of Applied Mathematics* 16, 1 (1958), 87–90.

[3] Jim Browne, Didier Dubois, Keith Rathmill, Suresh P. Sethi, and Kathryn E. Stecke. 1984. Classification of flexible manufacturing systems. *The FMS Magazine* 2, 2 (1984), 114–117. DOI: http://dx.doi.org/10.1023/A:1008062220281

[4] Frederico Busato and Nicola Bombieri. 2016. An Efficient Implementation of the Bellman-Ford Algorithm for Kepler GPU Architectures. *IEEE Transactions on Parallel and Distributed Systems* 27, 8 (Aug 2016), 2222–2233. DOI: http://dx.doi.org/10.1109/TPDS.2015.2485994

[5] Suresh Chand, Rodney Traub, and Reha Uzsoy. 1997. Rolling horizon procedures for the single machine deterministic total completion time scheduling problem with release dates. *Annals of Operations Research* 70, 0 (1997), 115–125. DOI: http://dx.doi.org/10.1023/A:1018961818782

[6] Jen-Shiang Chen, Jason Chao-Hsien Pan, and Chien-Kuang Wu. 2008. Hybrid tabu search for re-entrant permutation flow-shop scheduling problem. *Expert Systems with Applications* 34, 3 (2008), 1924–1930. DOI: http://dx.doi.org/10.1016/j.eswa.2007.02.027

[7] Ebru Demirkol and Reha Uzsoy. 2000. Decomposition methods for reentrant flow shops with sequence-dependent setup times. *Journal of Scheduling* 3, 3 (2000), 155–177. DOI: http://dx.doi.org/10.1002/(SICI)1099-1425(200005/06)3:3⟨155::AID-JOS39⟩3.0.CO;2-E

[8] Lester R. Ford Jr. 1956. *Network Flow Theory.* Rand Corp, Santa Monica, CA, USA.

[9] Józef Grabowski, Ewa Skubalska, and Czesław Smutnicki. 1983. On Flow Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness. *Journal of Operational Research Society* 34, 7 (1983), 615–620. DOI: http://dx.doi.org/10.1057/jors.1983.142

[10] Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5 (1979), 287–326. DOI: http://dx.doi.org/10.1016/S0167-5060(08)70356-X

[11] BongJoo Jeong and Yeong-Dae Kim. 2014. Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times. *Computers and Operations Research* 47 (2014), 72–80. DOI: http://dx.doi.org/10.1016/j.cor.2014.02.002

[12] Caixia Jing, Wanzhen Huang, and Guochun Tang. 2011. Minimizing total completion time for re-entrant flow shop scheduling problems. *Theoretical Computer Science* 412, 48 (2011), 6712–6719. DOI: http://dx.doi.org/10.1016/j.tcs.2011.08.030

[13] Jitti Jungwattanakit, Manop Reodecha, Paveena Chaovalitwongse, and Frank Werner. 2008. Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *International Journal of Advanced Manufacturing Technology* 37, 3 (2008), 354–370. DOI: http://dx.doi.org/10.1007/s00170-007-0977-0

[14] Wieslaw Kubiak, Sheldon XC Lou, and Yingmeng Wang. 1996. Mean flow time minimization in reentrant job shops with a hub. *Operations Research* 44, 5 (1996), 764–776. DOI: http://dx.doi.org/10.1287/opre.44.5.764

[15] Jason Chao-Hsien Pan and Jen-Shiang Chen. 2003. Minimizing makespan in re-entrant permutation flow-shops. *Journal of the Operational Research Society* 54, 6 (2003), 642–653. DOI: http://dx.doi.org/10.1057/palgrave.jors.2601556

[16] Robert Sedgewick and Kevin Wayne. 2011. *Algorithms, 4th Edition.* Addison-Wesley.

[17] John P. Shewchuk and Colin L. Moodie. 1998. Definition and Classification of Manufacturing Flexibility Types and Measures. *International Journal of Flexible Manufacturing Systems* 10, 4 (1998), 325–349. DOI: http://dx.doi.org/10.1023/A:1008062220281

[18] Hamid Shojaei, Twan Basten, Marc Geilen, and Azadeh Davoodi. 2013. A Fast and Scalable Multidimensional Multiple-choice Knapsack Heuristic. *ACM Trans. Des. Autom. Electron. Syst.* 18, 4, Article 51 (Oct. 2013), 32 pages. DOI: http://dx.doi.org/10.1145/2541012.2541014

[19] Joost van Pinxten, Marc Geilen, Twan Basten, Umar Waqas, and Lou Somers. 2016. Online heuristic for the Multi-Objective Generalized traveling salesman problem. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016.* 822–825.

[20] Umar Waqas, Marc Geilen, Jack Kandelaars, Lou Somers, Twan Basten, Sander Stuijk, Patrick Vestjens, and Henk Corporaal. 2015. A Re-entrant Flowshop Heuristic for Online Scheduling of the Paper Path in a Large Scale Printer. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015.* 573–578. DOI: http://dx.doi.org/10.7873/DATE.2015.0519

[21] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm.* Technical Report. Swiss Federal Institute of Technology (ETH) Zurich.

Received March 2017; revised June 2017; accepted June 2017