An efficient heuristic for adaptive production scheduling and control in one-of-a-kind production

Wei Li[a], Barrie R. Nault[b], Deyi Xue[a], and Yiliu Tu[a*]

[a] *Department of Mechanical and Manufacturing Engineering, University of Calgary, Calgary, Alberta, Canada T2N 1N4;*
[b] *Management Information Systems Area, Haskayne School of Business, University of Calgary, Calgary, Alberta, Canada T2N 1N4;*

Abstract

Even though research in flow shop production scheduling has been carried out for many decades, there is still a gap between research and application – especially in manufacturing paradigms such as one-of-a-kind production (OKP) that intensely challenges real time adaptive production scheduling and control. Indeed, many of the most popular heuristics continue to use Johnson's algorithm (1954) as their core. This paper presents a state space (SS) heuristic, integrated with a closed-loop feedback control structure, to achieve adaptive production scheduling and control in OKP. Our SS heuristic, because of its simplicity and computational efficiency, has the potential to become a core heuristic. Through a series of case studies, including an industrial implementation in OKP, our SS-based production scheduling and control system demonstrates significant potential to improve production efficiency.

Key words: Flow shop scheduling; Adaptive production control; Petri nets; Simulation

1. Introduction

As a typical manufacturing paradigm, one-of-a-kind production (OKP) challenges production scheduling and control differently than mass production. High throughput in OKP is an extreme example of mass customization, which is one of the important strategies in the current economy [1] where the objective is to maximize the customer satisfaction by producing highly customized products with near mass production efficiency. OKP is intensely customer focused such that every product is based on specific customer requirements, and products differ on matters of col-

*Corresponding author. Tel: 1-403-220-4142, Fax: 1-403-282-8406
E-mail address: paultu@ucalgary.ca

ors, shapes, dimensions, functionalities, materials, processing times, and so on. Consequently, a product that is produced on an OKP flow line is rarely repeated [2], although some processes in the production of similar kinds of products can be repeated. Moreover, unexpected disturbances frequently and randomly happen to affect the daily production on OKP shop floors, such as job insertion/cancellation, machine breakdown/operator absence, and variations in processing times. Thus, OKP companies use mixed-product production on a flow line to improve production efficiency [3], [4], and they have to adaptively schedule and control production online.

If a customer order in OKP is viewed as a project, to schedule the production of products in the customer order in OKP is very similar to scheduling a project. In fact, to schedule production in OKP is concurrent engineering, including product design, process planning, resource allocation, and finally the production schedule. Through this concurrent engineering effort, a combined engineering file for a product, including bill of materials (BOM), bill of operations (BOO), and resource constraints, is generated. This file is referred to as a product production structure (PPS) [5]. After the generation of a PPS for an OKP product, the processing times of each operation in the PPS are quoted based on the previous production of similar products. After all the PPSs for a batch of OKP products are determined and the processing times of all the operations are quoted, heuristics are needed to finally sequence the products in the batch to minimize the makespan. The state space (SS) heuristic that we present in this paper is typically for flow shop scheduling. In project scheduling, the emphasis is placed on how to allocate scarce resources to dependent activities or operations of a project to control the budget or minimize the duration of the project [6], [7]. These dependent activities or operations are normally arranged in a hierarchy, commonly known as a precedence diagram. If the sequence of a series of projects needs to be adaptively adjusted in order to minimize the duration to complete all projects, the SS heuristic may be applied.

Currently, OKP management primarily uses priority dispatching rules (PDRs) to deal with disturbances. It is fast and simple to use PDRs to control production online, but PDRs depend heavily on the configuration of shop floors, characteristics of jobs, and scheduling objectives [8], and there is no specific PDR that clearly dominates the others [9]. Moreover, the performance of PDRs is poor on some scheduling objectives [10], and it is especially inconsistent when a processing constraint changes [11]. Consequently, there is a considerable difference between the scheduled and actual production progress [12]: when unexpected changes occur and PDRs are

used to adaptively control production, production often runs in a chaotic or an "ad hoc fire fighting" manner [13], [14].

Indeed, due to dynamic disturbances, OKP has to be adaptively scheduled and controlled [2], [14]. When adaptive production control is taken into consideration, a closed-loop control structure is necessary and an efficient heuristic is critical. We propose a state space (SS) heuristic to support a computer-aided production scheduling and control system. We compare the optimality of our SS heuristic in terms of minimizing the maximum completion time, to the CDS heuristic [15] and to the NIS heuristic [16] using case studies based on well-accepted benchmarks, for both traditional flow shop (TFS) and hybrid flow shop (HFS) problems under no pre-emption or no wait processing constraints. Both the CDS and NIS heuristics use Johnson's algorithm as their core. In addition to its self-contained performance, we believe that our SS heuristic – because of its simplicity and computational efficiency – has the potential to become a core heuristic.

We find that across our different case studies the SS heuristic outperforms the CDS and NIS heuristics. In addition, in a real industrial application at an OKP company, Gienow Windows and Doors, our production scheduling and control system based on the SS heuristic reduced the company's original scheduling period using PDRs by an order of magnitude from three days to two hours, providing the company significant flexibility and competitiveness.

The rest of this paper is organized as follows. Section 2 gives a brief literature review. Section 3 introduces the SS heuristic. Section 4 presents the scheduling system and a closed-loop control structure for adaptive control in OKP. Section 5 gives the results from our case studies on TFS and HFS problems under no pre-emption or no wait processing constraints, operator absence disturbances, and in an industrial setting. We also provide a possible extension of our SS heuristic. Finally, Section 6 draws conclusions and proposes future work.

2. Literature review

Research in production scheduling has been carried out for many decades, and there are numerous scheduling methods developed in the literature. In this section, we briefly review flow shop production scheduling methods, and discuss the requirements of heuristics for adaptive production control.

Scheduling is a decision making process of allocating resources to jobs over time to optimize one or more objectives. According to [17], one type of flow shop consists of m machines in series, and each job has to be processed on each one of m machines in a single direction, which

means first on machine 1, then machine 2, and so on. This is typically called a traditional flow shop (TFS). Another type of flow shop where there are *S* stages in series with a number of machines/operators in parallel in each stage is a flexible flow shop or hybrid flow shop (HFS). In addition to the difference in flow shop configurations, processing constraints are also different for TFS and HFS. For TFS, if the first in first out (*FIFO*) discipline is applied to jobs in work-in-process (WIP) inventories, then it becomes a no pre-emption or permutation (*prmu*) flow shop problem. For HFS, if the first come first serve (*FCFS*) discipline is applied, then it is still a no pre-emption flow shop problem but the output sequence from each stage may change. Another processing constraint could be no wait (*nwt*), that is, jobs are not allowed to wait between two machines or stages, which also means there is no intermediate storage. The most common objective of flow shop scheduling is to minimize the maximum completion time or makespan, that is, min (C_{max}). Following the popular three parameter notation, $\alpha/\beta/\gamma$, the above problems can be expressed as $Fm/prmu/C_{max}$ for *m* machine TFS problems with no pre-emption constraint to minimize makespan, $Fm/nwt/C_{max}$ for *m* machine TFS problems with no wait constraint to minimize makespan, $FFs/FCFS/C_{max}$, for *S* stage HFS problems with *FCFS* constraint to minimize makespan, or $FFs/nwt/C_{max}$ for *S* stage HFS problems with no wait constraint to minimize makespan.

Gupta and Stafford [18] chronologically reviewed flow shop scheduling research in the past five decades since the classic Johnson's algorithm in 1954. They found that the emergence of NP-completeness theory in the third decade (1975 – 1984) profoundly impacted the direction of research in flow shop scheduling. That is why heuristics are required to solve large problems. HFS problems emerged in the fourth decade (1985 – 1994), and various artificial intelligence based heuristics were proposed then. The fifth decade (1995 – 2004) witnessed the proliferation of various flow shop problems, objective functions, and solution approaches. Although flow shop scheduling has been researched for more than fifty years, there remains a large gap between theoretical research and industrial applications [18].

Framinan et al. [19] proposed a general framework for the development of heuristics which consists of three phases: index development, solution construction, and solution improvement. Phase I, index development, means that jobs are arranged according to a certain property based on processing times. For example, Campbell et al. [15] extended Johnson's algorithm and proposed the CDS heuristic for an *n*-job *m*-machine TFS problem to min (C_{max}). The CDS heuristic

using Johnson's algorithm to arrange jobs is as follows. If there is a counter (*Ctr*) pointing to a machine *j*, then for each job *i* (*i* = 1…*n*) the sum of processing times on the first *Ctr* machines is regarded as its processing time on virtual machine 1, and the sum of processing times on the rest *m* – *Ctr* machines as its processing time on virtual machine 2. Then apply Johnson's algorithm to this virtual 2-machine flow shop problem to get a sequence. As *Ctr* changes from 1 to *m* – 1, *m* – 1 sequences are generated, and the one with the minimum makespan is the final solution. In phase II, solution construction, a solution is constructed by a recursive procedure, trying to insert one or more unscheduled jobs into a specific position of a partial sequence until the final schedule is completed. NEH [20] is a typical phase II heuristic for an *n*-job *m*-machine TFS problem to min (C_{max}). The NEH procedure is as follows. First, for each job, NEH sums the processing times on all of *m* machines, and then arranges these sums in a non-ascending order. Second, NEH schedules the first two jobs to get a partial sequence, and then inserts the third job into three possible positions to get another partial sequence, and so on. Finally, NEH inserts the last job into *n* possible positions, and then determines the final schedule. In phase III, solution improvement, there are two main characteristics. The first is that there must be an initial schedule, and the second is, after using artificial intelligence techniques, the quality of solution is better than the initial schedule. For the future development of heuristics, Framinan et al. [19] clearly stated the importance of heuristic development in phase I, index development, should not be underestimated, as it is required for the other two phases.

In a case study including 19 constructive heuristics for $Fm/prmu/C_{max}$ problems, Ruiz and Maroto [10] concluded that the NEH heuristic is best, the CDS heuristic is 8th, and two PDRs (LPT and SPT rules) are the worst. However, the CDS heuristic has the simplest computational complexity among the first 8 heuristics, $O(m^2n+mnlogn)$. Moreover, King and Spachis [11] did case studies of 5 PDRs and the CDS heuristic for two different TFS problems, $Fm/prmu/C_{max}$ and $Fm/nwt/C_{max}$. They concluded that the CDS heuristic and LWBJD (least weighted between jobs delay) rule work best for $Fm/prmu/C_{max}$ problems and MLSS (maximum left shift savings) rule works best for $Fm/nwt/C_{max}$ problems, but no single method works best for both $Fm/prmu/C_{max}$ and $Fm/nwt/C_{max}$ problems.

Compared with TFS, the literature on HFS is still scarce [21], [22]. According to Botta-Genoulaz [23], the CDS heuristic is the best of 6 heuristics, including the NEH heuristic, for HFS problems. The problem that Botta-Genoulaz solved is an *n*-job *S*-stage HFS problem to

minimize the maximum lateness, min (L_{max}), which was cleverly converted to an n-job S+1-stage HFS problem to min (C_{max}). The processing time of job i in stage S+1 is calculated by $p_{iS+1} = D_{max} - d_i$, $i = 1 \ldots n$, where $D_{max} = \max [d_k]$, and d_k is the due date for job k, $k = 1 \ldots n$. Although the CDS heuristic was constructed for TFS problems to min (C_{max}), when applying the CDS heuristic to HFS problems, Botta-Genoulaz also converts the processing times, $p'_{is} = p_{is} / OPTR_s$, $s = 1 \ldots S$+1, where p_{is} is the original processing time of job i in stage s, and $OPTR_s$ is the number of operators/machines assigned to stage s. It is reasonable to compare our SS heuristic with the CDS heuristic for $FFs/FCFS/C_{max}$ problems, which is computationally efficient and itself is based on Johnson's algorithm, because the basis of Botta-Genoulaz's conclusion is to min (C_{max}) for n-job S+1-stage HFS problems.

For HFS problems with identical parallel machines and with a no wait constraint, $FFs/nwt/C_{max}$, Thornton and Hunsucker [16] proposed a no intermediate storage (NIS) heuristic, which works best among the CDS heuristic, LPT and SPT rules, and a heuristic of random sequence generation. Like the CDS heuristic, the NIS heuristic also applies Johnson's algorithm and uses a filter concept to convert an $FFs/nwt/C_{max}$ problem to a 2-machine problem. The stages before the filter are regarded as virtual machine 1, the stages after the filter are regarded as virtual machine 2, and the stages that are covered by the filter are omitted. The filter goes from stage 2 to stage $S - 1$, and the width of filter changes from 1 to $S - 2$, which means filtering out processing times in 1 stage or $S - 2$ stages at a time. In total, there are $1 + (S - 1) \times (S - 2) / 2$ sequences generated by the NIS heuristic and the one with the minimum makespan is the final schedule, from which we can see that NIS is not very computational efficient. Again, because NIS is based on Johnson's algorithm, comparing our SS heuristic to NIS is reasonable.

From our point of view there are three main criteria on which to evaluate heuristics: optimality, computational complexity, and flexibility. Usually optimality is used to evaluate the performance of a heuristic for offline production scheduling. However, when adaptive production control is taken into consideration, the computational complexity becomes critical. That is why some artificial intelligence based heuristics are not suitable for adaptive control, although they can get better solutions. Another criterion is the flexibility of a heuristic, that is, whether a heuristic can deal with disturbances. Of course, different situations have different requirements for optimality, computational complexity, and flexibility of a heuristic. There is inevitably a trade-off among those three criteria, and the selection of heuristic(s) for adaptive control depends on

the specifics of different situations such as the value of optimality as compared to close-to-optimal scheduling, as well as the type and volume of disturbances that underlies the requirements of response time.

For example, given two heuristics, H_1 and H_2, two types of disturbances, D_1 and D_2, and a maximum response time for online rescheduling t, some characteristics of heuristics are shown in Table 1 for a given number of jobs.

[Table 1]

When disturbance D_1 happens to production, the selection of a heuristic between H_1 and H_2 depends on optimality and computational complexity, because both heuristics are flexible enough to deal with D_1. If $T_{11} \leq t$ and $T_{21} \leq t$, the selection of a heuristic depends only on optimality, which means a mutual comparison. Otherwise, the selection depends on computational complexity. The mutual comparison is necessary especially when no benchmarks are available to evaluate optimality of heuristics separately for large problems. When disturbance D_2 happens, we can only choose heuristic H_2 if $T_{22} \leq t$. However, if $T_{22} > t$, and such maximum response time t cannot be relaxed, neither heuristic H_2 nor heuristic H_1 can be selected.

In general, the flexibility of a heuristic must meet the requirement of specific scheduling problems. After that, computational complexity is a critical selection criterion for online rescheduling (i.e., adaptive production control), whereas optimality is a dominant selection criterion for offline scheduling.

3. The state space (SS) heuristic

The SS heuristic minimizes the maximum completion time (or makespan) for HFS problems, in which the *FCFS* discipline is applied and the capacity of WIP inventories is limited. Because there are multiple operators in each stage of a flow line, SS heuristic also maximizes the utilization of the flow line, max (*Util*).

There are two main concepts applied in our SS heuristic which we introduce in subsection 3.1, and then the main steps of the SS heuristic are introduced in subsection 3.2. Finally, the computational complexity of the SS heuristic is briefly analyzed in subsection 3.3.

3.1. Two concepts in the SS heuristic

3.1.1. The state space concept

Consider a hybrid flow line with 3 work stages and 2 operators in each stage (Fig. 1). The operators in each stage follow a *FCFS* rule, that is, the operator who is available first for the next job should serve immediately in the stage. Then there is a next available time of each stage, A_s, where $A_s = \min(a_{sk})$, for $k = 1 \ldots OPTR_s$, in which a_{sk} is the next available time of operator k in stage s, and $OPTR_s$ is the number of operators allocated to stage s. For an S-stage flow line, there are $S - 1$ time differences between stage available times. In the example above, there are two differences of the next available times, $A_2 - A_1$, and $A_3 - A_2$. If we regard such a difference of the next available times as a space, $SPACE_s = A_{s+1} - A_s$ for $s = 1 \ldots S\text{-}1$, then $SPACE_s$ is a time period available for stage s to finish a job without causing an operator in stage s+1 to be idle. If the completion time of job i in stage s is larger than the next available time of stage s+1, then such a job causes stage s+1 to be idle for $IDLE_{is} = c_{is} - A_{s+1}$ time units, where c_{is} is the completion time of job i in stage s, $c_{is} = \max(A_s, c_{is-1}) + p_{is}$, where p_{is} is the processing time of job i in stage s. The completion time c_{is} depends on the maximum of job i's completion time from the previous stage $s - 1$, c_{is-1}, and the next available time of current stage s, A_s. If the completion time of job i in stage s is smaller than the next available time of stage s+1, then there are two possibilities depending on whether WIP is full. If the WIP inventory, WIP_s, is full, then a delay happens to operator k who processed job i in stage s, $DELAY_{is} = A_{s+1} - c_{is}$. Such a delay means that, after finishing job i, operator k in stage s has to hold it in hand for $DELAY_{is}$ time units until there is a vacancy in WIP_s. Therefore, the next available time of operator k in stage s is delayed. Alternatively, if WIP_s is not full, job i goes into inventory, and there is no *IDLE* and no *DELAY*.

[Fig. 1]

The main idea of SS is to find a job that fits $S - 1$ spaces, without causing *IDLE* or *DELAY* time. After a job i is processed on a line, the state of the line, i.e., the next available time of stages, is changed, and the space is changed accordingly. It is clear that greater *IDLE* and *DELAY* are not good for production if the objectives are to min (C_{max}) and max (*Util*), while greater *SPACE* is good for production to some extent. Therefore, in SS, job i is chosen according to its performance on *IDLE*, *DELAY*, and *SPACE*.

From the foregoing description of SS, we can see that *IDLE* and *DELAY* are evaluated according to job i and stage s, but *SPACE* is only evaluated by stage s. To make *SPACE* both job and stage dependant, there could be two ways to model *SPACE*. One model is $SPACE_{is} = c_{is+1} - A_s$, for $s = 1 \ldots S\text{-}1$, which means using the difference between the completion time of job i in

stage s+1 and the next available time of stage s as the *SPACE* created by job i for stage s to process the next job. The other model is $SPACE_{is} = p_{is+1}$, for $s = 1 \ldots S\text{-}1$, which means using the processing time of job i in stage s+1 as *SPACE*. In our current version of the SS heuristic, we use the latter model of *SPACE* because there is one less calculation for each iteration, increasing the computation speed for adaptive control. However, we illustrate the alternative model of *SPACE*, $SPACE_{is} = c_{is+1} - A_s$, in a case study in Section 5 to show the flexibility of the SS concept.

3.1.2. The lever concept

From our previous research on TFS problems, we found that the lever concept is suitable for flow shop production [24], which means that *IDLE* (or *DELAY*) in an earlier stage is worse for min (C_{max}) objective than in a later stage. Consider a lever where force F takes effect and causes a torque of $F \times L$, where F is the unit of force and L is the length of force arm. We model an S stage flow line as a lever, and $IDLE_{is}$ caused by job i, or $DELAY_{is}$, has a torque effect manifested as $IDLE_{is} \times LVR\_IDLE_s$ or $DELAY_{is} \times LVR\_DELAY_s$, where LVR_IDLE_s or LVR_DELAY_s is the length of arm for $IDLE_{is}$ or $DELAY_{is}$ respectively.

The lever concept for *IDLE* (LVR_IDLE_s) in SS is shown in Fig. 2. For an S-stage flow line, a job could cause at most $S - 1$ times of *IDLE*, since no *IDLE* is caused in stage 1 and an *IDLE* takes effect in the next stage. Therefore, the fulcrum of a lever for *IDLE* is set between stage $S - 1$ and stage S, and the length of arm for an *IDLE* caused by stage s in stage s+1 is $LVR\_IDLE_s = S - s$, for $s = 1 \ldots S\text{-}1$.

[Fig. 2]

A lever concept for *DELAY* (LVR_DELAY_s) in SS is shown in Fig. 3. Like the number of possible *IDLE*s, there could be $S - 1$ *DELAY*s, since no *DELAY* is caused in stage S. But a *DELAY* takes effect in current stage s, whereas *IDLE* is for the next stage. Therefore, one unit of *DELAY* in stage s should be worse for min (C_{max}) objective than one unit of *IDLE* in stage s. So the length of arm for a *DELAY* is $LVR\_DELAY_s = S - s + 1$, for $s = 1 \ldots S\text{-}1$. The fulcrum of a lever for *DELAY* is set in stage S.

[Fig. 3]

There is also a lever concept for *SPACE* (LVR_SPACE_s) in SS, shown in Fig. 4. The length of force arm for a space is $LVR\_SPACE_s = s$, for $s = 1 \ldots S\text{-}1$. The fulcrum of a lever for *SPACE* is set between stage 1 and stage 2, which means *SPACE* in a later stage is better for min (C_{max}) than in an earlier stage.

[Fig. 4]

Therefore, all *SPACE*s, *IDLE*s, and *DELAY*s are converted to torques, that is, $SPACE'_{is} = p_{is+1} \times LVR\_SPACE_s$, $IDLE'_{is} = IDLE_{is} \times LVR\_IDLE_s$, and $DELAY'_{is} = DELAY_{is} \times LVR\_DELAY_s$. The job selection scheme is then

$$\max_{1 \le i \le n} \left[\sum_{s=1}^{S-1} SPACE'_{is} - \left(\sum_{s=1}^{S-1} IDLE'_{is} + \sum_{s=1}^{S-1} DELAY'_{is} \right) \right],$$

that is, to select one out of n unscheduled jobs with the maximum torque difference between *SPACE'*s and *IDLE'*s + *DELAY'*s.

Take the process of job i on a 4-stage flow line as an example. The operation of job i in stage 1 would cause a 3-time unit idle to stage 2. The operation of such job in stage 2 would cause 0-time unit idle to stage 3, but a 2-time unit delay in stage 2, and the operation of such job in stage 3 would cause a 1-time unit idle to stage 4. Accordingly, such job would create a 2-time unit space between stage 1 and 2, a 6-time unit space between stage 2 and 3, and a 3-time unit space between stage 3 and 4. Thus, $\sum_{s=1}^{3} IDLE'_{is}$ created by job i equals to 3×3 + 0×2 + 1×1 = 10, $\sum_{s=1}^{3} DELAY'_{is}$ equals to 0×4 + 2×3 + 0×2 = 6, and $\sum_{s=1}^{3} SPACE'_{s}$ equals to 2×1 + 6×2 + 3×3 = 23.

3.2. The steps to achieve the SS heuristic

3.2.1. Initial job selection

The main idea of SS is to select a successive job according to the state space generated by previous jobs in a sequence. Therefore, it is important to select initial jobs to create an initial state space, which affects the selection of successive jobs and ultimately the schedule performance.

Two items should be taken into consideration for initial job selection in SS. One is the number of initial jobs – which is straightforward, and the other is the initial job selection scheme. The number of initial jobs is set as min ($OPTR_s$, for $s = 1 \ldots S$). The reason is that if the number of initial jobs is smaller than min ($OPTR_s$), then the state (the first available time of a stage) is zero since all operators in all stages are available at time zero; if the number is greater than min ($OPTR_s$), then the number of jobs (initial job number – min ($OPTR_s$)) are not selected by the state space concept.

For initial job selection scheme, five $1 \times S$ vectors are introduced as follows:

*Vector*_1 = [0];

*Vector*_3 = [APT_s], where $APT_s = \left(\sum_{i=1}^{N} p_{i,s} \right) \Big/ n$, i.e., the average processing time of stage *s*;

*Vector*_5 = [max (p_{is}), i = 1…n] for s = 1…S, i.e., the maximum processing time in each stage;

*Vector*_2 = *Vector*_3 / 2;

*Vector*_4 = *Vector*_3 + [*Vector*_5 – *Vector*_3] / 2.

Take an *n*-job 4-stage production problem as an example. *Vector*_1 should be [0, 0, 0, 0], if *Vector*_3 = [4, 6, 8, 10], and *Vector*_5 = [10, 12, 16, 20], then *Vector*_2 = [2, 3, 4, 5] and *Vector*_5 = [7, 9, 12, 15].

The initial number of jobs are selected according to $\min\left(\sum_{s=1}^{S} \left(\left| p_{is} - Vector\_v(s) \right| \right) \right)$ for i = 1…n, which means the minimum absolute difference between one job's processing times and the vector.

3.2.2. Steps

The following steps represent the generic programming logic of the SS heuristic.

Step 1: Determine the number of operators in each stage, i.e., $OPTR_s$. (1a): Calculate n and S. (1b): Set an expected throughput rate, r, which means in every r time units a job is expected to be finished in a stage. (1c): $OPTR_s$ = Roundup (APT_s / r). (1d): Set the start time of every operator to 0. (1e): Put all of n jobs into a candidate pool. (1f): Set an output sequence to be a 1 × n zero vector, *Sequence_v*.

Step 2: Set the capacity of each of S – 1 WIP inventories. The capacity of each WIP inventory could be different.

Step 3: Calculate five vectors for initial job selection.

Step 4: FOR v = 1:5. This is an iteration loop to select initial jobs according to one of five vectors, i.e., *Vector_v*.

Step 5: Select min ($OPTR_s$, for s = 1…S) number of jobs according to *Vector_v* by the equation $\min\left(\sum_{s=1}^{S} \left(\left| p_{is} - Vector\_v(s) \right| \right) \right)$ for i = 1…n. Then put selected jobs into a *Sequence_v* and eliminate them from the candidate pool. Calculate the next available time of each

operator, and then the next available time of each stage, namely *STATE*, and WIP inventory status, namely *WIP_Status*, which is now a $1\times(S-1)$ zero vector.

Step 6: FOR i = min ($OPTR_s$) + 1: n. This is an iteration loop to sequence the rest of n – min ($OPTR_s$) jobs in the pool.

Step 7: According to *STATE* and *WIP_Status*, calculate $IDLE'_{is}$, $DELAY'_{is}$, and $SPACE'_{is}$.

Step 8: Select job i according to $\max_i\left[\sum_{s=1}^{S-1} SPACE'_{is} - \left(\sum_{s=1}^{S-1} IDLE'_{is} + \sum_{s=1}^{S-1} DELAY'_{is}\right)\right]$, and then put such job number into *Sequence_v* and eliminate it from the candidate pool.

Step 9: Calculate intermediate completion time of a partial schedule *Sequence_v*, update *WIP_Status*, and update *STATE*.

Step 10: END i. Calculate the utilization of a line. (10a): Calculate utilization of each stage first, $Util_s = (\sum_{i=1}^{n} p_{is} / OPTR_s) / (c_{nks} - c_{1k's\text{-}1})$, $c_{1k0} = 0$, $s = 1\ldots S$, in which c_{nks} is the completion time of the last finished job in stage s, and $c_{1k's\text{-}1}$ is the start time of the first job in stage s, i.e., the completion time of the first finished job in stage s-1. (10b): Calculate the average utilization of each stage, i.e., the utilization of a line, $Util$ = average ($Util_s$), $s = 1\ldots S$.

Step 11: END v. Output each of five sequences and related makespan and utilization, and the minimum makespan and the maximum utilization are regarded as the final performance of SS.

3.3. The computational complexity of the SS heuristic

The computational complexity of a heuristic is the highest order of operations for a specific problem. For an n-job S-stage HFS problem the computational complexity of our SS heuristic consists of two parts: job selection and the makespan calculation. The main scheme of SS heuristic to construct a job sequence is to select the next successive job according to the current state of the flow line. This scheme is for both offline scheduling and online adaptive control. Makespan calculation is to calculate the current state of the flow line, but it is carried out five times in the

SS heuristic for offline scheduling, since there are five sequences generated by the SS heuristic, although only one of them will be chosen as the final production schedule.

3.3.1. Computational complexity of job selection

If the state of the flow line is known, then to select one out of n unscheduled jobs takes $S \times n$ operations, which means the computational complexity for adaptive control is $O(Sn)$. As n decreases from n to 1, the computational complexity of overall selection of n jobs is $O(Sn^2)$. Although there are five sequences generated by the SS heuristic, the computational complexity of job selection of SS heuristic is not $O(5Sn^2)$, because only the highest order of operations is counted in computational complexity and 5 is a constant.

3.3.2. Computational complexity of the makespan calculation

An n-job and S-stage HFS problem can be modelled by a 2-dimension matrix, where the row dimension is for n jobs, and the column dimension is for S stages. The makespan calculation could be carried out along the column dimension. It means that, if the input sequence of n jobs of stage 1 is known, then the output sequence of n jobs of stage 1, which is also the input sequence of stage 2, can be calculated; the output sequence is in a non-descending order of completion times of n jobs; and then the output sequence of stage 2 can be calculated, and so on, finally the output sequence of stage S can be calculated. However, the capacities of WIP inventories are limited, which means the completion times of jobs in stage s are constrained by the next available times of operators in stage s+1. For example, when calculating the output sequence of stage s, if a job i's completion time in stage s causes the WIP_s inventory to be overloaded, which means at that time the WIP_s inventory is full and there is no operator available in stage s+1 to process a job in the WIP_s inventory, then a *DELAY* happens to such job i. This *DELAY* means the job i's completion time is delayed to a later time, and so is the next available time of operator k, who processes the job i in stage s. Consequently, the *DELAY* affects the completion times of all jobs following job i in stage s, and the completion times in the previous stage need to be checked because of the limitation on WIP inventories. In an extreme situation, when a *DELAY* happens in stage S-1, the completion times of jobs in all previous stages have to be recalculated. Because of the many recalculations, it is time consuming to calculate makespan along the column dimension when the capacity of each WIP inventory is small. Therefore, the makespan calculation should be done along the row dimension, i.e., calculated by jobs and not by stages.

For the makespan calculation, as n increases from 1 to n, the computational complexity is also $O(Sn^2)$, although makespan calculation is carried out five times. Therefore, the overall computational complexity of the SS heuristic is $O(Sn^2)$.

4. A close-loop production scheduling and control structure

Our proposed production scheduling and control system is based on a closed-loop control structure, which is common in adaptive control theory. The system, as illustrated in Fig. 5, consists of our SS heuristic and a simulation model that is called temporized hierarchical object-oriented coloured Petri nets with changeable structure (THOCPN-CS) [25]. High customization and dynamic uncertainties in OKP demand for a great effort on a simulation model. Simultaneously, adaptive production control demands for prompt solutions in time. Therefore, the unique feature of the THOCPN-CS simulation model, changeable structure, makes it easy and flexible to simulate frequent changes in OKP for adaptive production control. Steps to achieve adaptive production scheduling and control in OKP are summarized as follows.

[Fig. 5]

Step 1: Manually assign possible manufacturing resources (e.g., operators/machines) to each stage, and hence form a task-resource matrix (TRM) with jobs.

Step 2. Schedule the jobs by the SS heuristic.

Step 3. The THOCPN-CS model will simulate the production, and identify the bottleneck stage(s). Human schedulers may re-allocate operators/machines in stages accordingly, to smooth the production flow.

Step 4. Re-schedule the jobs by the SS heuristic.

Step 5. Repeat Steps 3 and 4 in the offline production scheduling phase until a satisfactory production schedule is obtained. This production schedule contains a job sequence and a number of operators/machines in each stage, nearly balancing the utilization of a production line. In the adaptive production control phase, this step may be omitted, depending on specific production requirements.

Step 6. Deliver the production schedule to the shop floor and switch the control loop from the simulation model to the shop floor.

Step 7. If any disturbance occurs on a shop floor, switch the control loop back to the simulation model, and go back to Step 3 if operators/machines re-allocation is necessary, or go back to Step 4.

Through iteratively repeating the above-mentioned steps, the production of an OKP shop floor can be adaptively scheduled and controlled to cope with disturbances. Various disturbances frequently happen to the OKP shop floor. Thus, the adaptive production control is a combination of users' knowledge and the heuristics' ability (optimality, computational complexity, and flexibility).

Having been implemented in an OKP company, Gienow Windows and Doors, our production scheduling and control system has reduced Gienow's original production scheduling period from three days to two hours. It means that the production schedule is fixed only for two hours, making the company more flexible in dealing with customer orders and improving its competitiveness.

5. Case studies

To test the performance of our SS heuristic, we carry out three kinds of case studies: SS performance as compared to other heuristics, SS performance with disturbances, and an industrial case study. For the performance of SS compared with other heuristics, we carry out four separate studies on different flow shop configurations: $Fm/prmu/C_{max}$, $Fm/nwt/C_{max}$, $FFs/FCFS/C_{max}$, and $FFs/nwt/C_{max}$. Taillard's benchmarks are used in case studies [26], because, to our best knowledge, such benchmarks are well designed and the most accepted for flow shop scheduling problems. In Taillard's benchmarks, the machine number ranges from 5 to 20, the job number ranges from 20 to 500, the processing time follows a uniform distribution from 1 to 99, and there are 120 instances in total. Although all these case studies are carried out based on Taillard's benchmarks, the result of $Fm/prmu/C_{max}$ is a comparison with best known upper bounds. For the remaining case studies on SS performance, the results are mutual comparisons, that is, the improvement of one heuristic over another.

To examine the performance of SS in the presence of disturbances, we examine the performance of SS with operator absence/machine breakdown under two definitions of SPACE, and using a simple optimization method together with SS.

In the third kind of case study we compare the SS heuristic with data provided by an industry partner, and measure the improvement due to SS. Finally, we show with a simple two-machine TFS example that the State Space concept can provide more solutions than Johnson's algorithm, and thus provide more opportunities as the core of a more elaborate heuristic.

5.1. Case studies on Taillard's benchmarks

5.1.1. Fm/prmu/C_{max}

For traditional permutation (or no pre-emption) flow shop scheduling problems, the deviation (*DEV*) from the best known upper bounds is used to evaluate the performance of a heuristic, where *DEV* = (*C_{max} of a heuristic – The upper bound*) ÷ (*The upper bound*) in percentage. The results of the deviation studies for CDS, NIS, SS, and a version of SS without the lever concept, SSnoLVR, heuristics are shown in Table 2.

[Table 2]

In Table 2, the column "Scale" means the size of problems. For example, 20*5 means 20-job 5-machine problems. The column "Instance" means the number of instances in each scale. Columns 3, 4, 5, and 6 represent the average deviation of each of the CDS, NIS, SS, and SSnoLVR heuristics respectively. For the total average deviation, the SS heuristic has the smallest deviation from Taillard's benchmarks, 8.11%, SSnoLVR heuristic ranks second at 8.80%, the NIS heuristic ranks third at 9.01%, and the CDS heuristic ranks last at 11.28%. These results show the strength of the SS heuristic and the value of the lever concept.

5.1.2. Fm/nwt/C_{max}

For traditional no wait flow shop scheduling problems, an improvement (*IMPR*) over the NIS heuristic is used to evaluate the performance of the CDS and SS heuristics based on Taillard's benchmarks, where *IMPR* = (*C_{max} of NIS – C_{max} of CDS or SS*) ÷ (*C_{max} of NIS*) in percentage is shown in Table 3.

[Table 3]

For *Fm*/*nwt*/C_{max} problems, the CDS heuristic performs 1.04% worse than the NIS heuristic. In contrast, the SS heuristic performs on average 2.27% better than the NIS heuristic. However, we recognize that for HFS problems the improvements of SS over NIS will shrink as the number of operators/machines in each stage increases. For example, if the number of operators/machines in each stage is the same as the number of jobs, then the C_{max} is fixed as max ($\sum_{s=1}^{S} p_{is}$) for i = 1…n, even for no wait or no pre-emption flow shop problems.

5.1.3. FFs/nwt/C_{max}

For hybrid no wait flow shop problems with identical parallel operators/machines in each stage, two operators/machines are assigned to each stage. The improvement of the CDS or SS

heuristics over the NIS heuristic is shown in Table 4. For such hybrid no wait flow shop problems with two operators/machines in each stage, the SS heuristic has a small improvement over the NIS heuristic, 0.39%, and the CDS heuristic still performs slightly worse, -1.33%.

[Table 4]

5.1.4. FFs/FCFS/C_{max}

For HFS problems with the *FCFS* discipline applied to jobs in WIP inventories, we set two variables. One is a throughput rate r = 31, used to calculate the number of operators in each stage, where $OPTR_s$ = Roundup (APT_s / r). The average processing time of each stage ranges from 30.75 to 64.40 for all of 120 instances in Taillard's benchmarks, therefore, $OPTR_s$ varies from 1 to 3 for each stage. Another variable is the capacity of WIP inventories. Different configurations of WIP inventories have different impacts on production [27]. For the ease of case study, we simply set each WIP inventory at the same WIP_s = 5, even though in theory each could be set to a different value. The calculation of processing times in CDS is $p'_{is} = p_{is} / OPTR_s$, s = 1…S [23]. The improvement (*IMPR*) of the SS heuristic over the CDS heuristic is used to evaluate performance for such scheduling problems for both min (C_{max}) and max (*Util*) objectives, where $IMPR_1$ = (C_{max} *of CDS* – C_{max} *of SS*) ÷ (C_{max} *of CDS*) in percentage, and $IMPR_2$ = (*Util of SS* – *Util of CDS*) ÷ (*Util of CDS*). The results are shown in Table 5. Over the 120 instances in Taillard's benchmarks, the SS heuristic has an average 1.16% improvement over the CDS heuristic on the min (C_{max}) objective and 3.96% on the max (*Util*) objective.

[Table 5]

5.1.5. Summary

In all four separate studies on different flow shop problems, on average, the SS heuristic performs better than the CDS and NIS heuristics, although worse for a few instances. However, the SS heuristic performs much better than the CDS and NIS heuristics for each of 30 large sized instances, in which the job number ≥ 200.

5.2. Case Studies on Operator Absence

To test the suitability of our SS heuristic to adaptive control, a case study of operator absence is carried out based on the data from Taillard's benchmarks. Modeling operator absence is the same as modeling machine breakdown. For this case study, we assume that, when a half of jobs are finished, one operator is absent in the middle stage of a flow line, specifically in stage 3, 6, or 11 according to the scale of instances in Taillard's benchmarks. For the remaining half of the

jobs, if the production is carried on according to the original schedule when such disturbances happen to the shop floor, then the completion time is recorded as *Original*. If adaptive control is applied, that is, using the SS heuristic to re-schedule the remaining jobs, then the completion time is recorded as *Adaptive*. The improvement of adaptive control over no adaptive control is used to evaluate the performance, which means (*Original – Adaptive*) ÷ (*Original*) in percentage.

To show the potential of our SS heuristic as a core heuristic, case studies on operator absence are carried out under the two definitions of *SPACE*, $SPACE_{is} = p_{is+1}$ and $SPACE_{is} = c_{is+1} - A_s$. As a third case study in this section we integrate a simple optimization method with the SS heuristic.

5.2.1. $SPACE_{is} = p_{is+1}$

The results are given in Table 6. As we see, adaptive control is slightly better than no adaptive control with a 0.10% improvement if $SPACE_{is} = p_{is+1}$.

[Table 6]

5.2.2. $SPACE_{is} = c_{is+1} - A_s$

The results are given in Table 7. We can see that adaptive control has a 2.02% improvement over no adaptive control.

[Table 7]

5.2.3. Integration with an optimization method

It is easy to integrate other optimization techniques into the State Space concept. For example, in the SS heuristic, there are two effects impacting the final production performance. Increased *SPACE* improves production, and *IDLE* and *DELAY* make production worse. If we introduce a weighting factor, α, into our SS heuristic that allows different weights for *SPACE* and for *IDLE* or *DELAY*, then we could sequence the jobs according to

$$\max_i \left[(1-\alpha) \times \sum_{s=1}^{S-1} SPACE'_{is} - \alpha \times \left(\sum_{s=1}^{S-1} IDLE'_{is} + \sum_{s=1}^{S-1} DELAY'_{is} \right) \right].$$

As α changes from 0 to 1 with increments of 0.1, the performance of the SS heuristic, in which $SPACE_{is} = p_{is+1}$, is shown in Table 8. The columns represent the performance of each α integrated with the SS heuristic. A weight α = 0.0 means no *IDLE* or *DELAY* is taken into consideration to sequence the jobs, and α = 1.0 means no *SPACE* is taken into consideration. The weighting factor α reveals the relationship between *SPACE*, *IDLE* and *DELAY* in adaptive control with Taillard's benchmarks: *SPACE* affects the production performance more than *IDLE* or *DELAY*, where α = 0.1 has the greatest improvement, 2.77%.

[Table 8]

5.3. An Industrial Case Study

To validate our SS heuristic in a real setting, an industrial case study has been carried out in Gienow Windows and Doors. This case consists of 1,396 jobs on a flow line with 5 stages for one-day production. These jobs are delivered to customers in 28 batches. Each batch of products is destined for customers in a given geographic area. Using data provided by Gienow and using the SS heuristic in its basic form, our heuristic produced the results shown in Table 9.

[Table 9]

As shown in the Table, Gienow used 42,300 time units to finish 1,396 jobs. The production of 1,396 jobs in 42,300 time units was achieved by Gienow's original schedule, which was generated by a production scheduler who has worked in Gienow for many years. Our SS heuristic can generate a new schedule, reducing 42,300 time units to 41,771, a 1.25% improvement in productivity. Such an improvement on daily production translates into the production of 17 additional products daily, or more than $5,000 revenue per day.

The SS heuristic was programmed in C++ language, and run on an Intel Pentium IV personal computer with a 3.20 GHz CPU and 1.00 GB RAM. The computation time of the SS heuristic to schedule 1,396 jobs on 5 machines was 70.67 seconds. Those 70.67 seconds are for offline production scheduling, generating and evaluating five sequences for all 1,396 jobs. However, for online adaptive production control, the computational complexity of SS heuristic is $O(Sn)$ (please refer to sub-sections 3.3.1 and 3.3.2). Therefore, if there are 1,396 jobs left unproduced when a disturbance occurs, it takes only 10.12 milliseconds ("the overall computation time"÷"5 sequences"÷ "1,396 jobs") for SS heuristic to select the next successive job, which is fast enough to respond to a disturbance on the shop floor.

Originally, it took three days for the experienced production scheduler to off-line schedule one-day production of customer orders in Gienow. This manual scheduling process made it difficult for the company to respond to changes of customer orders in time and at a low cost. Therefore, the company used a fixed three-day production scheduling policy, which means customers are not able to change or place an urgent order three days before production. With the proposed computer aided production scheduling and control system which is based on the SS heuristic, Gienow could change the production scheduling policy from three days to two hours, taking into

consideration of other constraints, e.g., raw material handling speed and capacity in the company, convenience of operators to switch between workstations or production lines.

5.4. An Extension of SS Heuristic

To further illustrate the State Space concept, we compare a scaled down version of SS with Johnson's algorithm for a two-machine TFS scheduling problem, $F2/prmu/C_{max}$. For $F2/prmu/C_{max}$ problems, the lever concept that is part of our SS heuristic has no effect on the job selection. This is because for such problems, there is no limit to the WIP inventory between machine 1 and 2, then no *DELAY* is taken into consideration. In addition, the length of force arm for *SPACE* or *IDLE* is equal to one for two-machine TFS scheduling problems. However, the State Space concept can yield different job sequences than Johnson's algorithm. A numerical example is provided in Table 10.

[Table 10]

Johnson's algorithm sequences the jobs according to the following scheme. If min (p_{a1}, p_{b2}) ≤ min (p_{a2}, p_{b1}), then job *a* should be processed earlier than job *b*. Therefore, for the example in Table 10, Johnson's algorithm generates a sequence of [Job 1, 3, 4, 2] with C_{max} = 62.

According to the State Space concept (again, not exactly the SS heuristic), and using Johnson's algorithm for initial job selection, we obtain two additional sequences, both of which have C_{max} = 62, and are different from the one generated by Johnson's algorithm: [Job 1, 2, 3, 4] and [Job 1, 4, 3, 2]. Thus, we can see that the State Space concept can yield different sequences than Johnson's algorithm with the same level of optimality, and hence can provide greater opportunities for improvement as the core of a more elaborate heuristic.

6. Conclusions

In this research we have presented the state space concept and developed the concept into a SS heuristic. As both a model (the state space concept) and a heuristic, we have shown that SS is useful for flow shop production scheduling and control when the number of machines or stages is greater than or equal to 3. In addition, we developed a lever concept to linearly scale measures of time availability (*SPACE*) and of idle and delay (*IDLE*, *DELAY*). Combining the state space and lever concepts in our SS heuristic, we showed that our heuristic has the computational complexity of $O(Sn^2)$. We also showed through case studies that the SS heuristic outperforms the most popular alternative heuristics (CDS, NIS) against Taillard's benchmarks both without and

with disturbances. We note that the CDS heuristic has a simpler computational complexity than our SS heuristic for scheduling itself, but if we take sequence evaluation into consideration, the CDS heuristic has the same computational complexity as the SS heuristic. The NIS heuristic has a higher computational complexity than our SS heuristic. To gauge the performance of the SS heuristic in an industrial setting, we obtained a dataset from a local OKP manufacturer, and found the schedule generated by our SS heuristic outperformed the manufacturer's schedule, and the schedule could be calculated in an order of magnitude less time.

According to the framework of heuristic development proposed by Framinan et al. [19], the SS heuristic is in the first of three phases (index development), a phase that is beneficial for heuristic development in other two phases (solution construction and improvement). Indeed, we also showed using a simple example, that as compared with Johnson's algorithm our state space concept can generate alternative schedules at the same level of optimality. Given that both the CDS and NIS heuristics have Johnson's algorithm as their core, the state space concept might be a good starting point to develop more elaborate heuristics for production scheduling with adaptive control.

The suitability of state space concept and the SS heuristic for adaptive control depends on its ability to handle disturbances and its computational complexity. Many disturbances can be handled by the state space concept. According to Pinedo [17], there can be three kinds of disturbances: job insertion/cancellation, operator absence/machine breakdown, and variation of processing times. For job insertion/cancellation, the processing times of jobs to be inserted or cancelled affect the next available times of operators in each stage, and consequently affect the time space between stages; similarly for operator absence/machine breakdown and variation of processing times. However, more research could improve the optimality of the SS heuristic beyond the implementation and case studies we describe. For computational complexity perspective, the SS heuristic is fast enough to generate a feasible solution for adaptive control because when a disturbance occurs, the computational complexity of the SS heuristic to choose the next job is only $O(Sn)$.

We recognize that implementing adaptive control solutions to deal with dynamic disturbances in manufacturing industries is difficult, and the gap between the theoretical research and industrial application is still large. More research is in need to reveal the relationship between jobs, stages, and operators in each stage. Better approximations for these relationships than the

linear version of the lever concept we use may be helpful, and this is one of the directions of our future research.

Acknowledgements

This research project has been funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada through NSERC Strategic Project Grant and NSERC Discovery Grants, and by Gienow Windows and Doors Co. Ltd., Calgary, Alberta, Canada.

References

[1] Blecker T, Friedrich G. (Eds.) Mass Customization: Challenges and Solutions. New York: Springer Science + Business Media, Inc., 2006

[2] Wortmann JC, Muntslag DR, Timmermans PJM. Customer-driven Manufacturing. London: Chapman & Hall, 1997.

[3] Dean PR, Tu YL, Xue D. A framework for generating product production information for mass customization. Int. J. Adv. Manuf. Tech., 2008; 38(11-12): 1244-1259.

[4] Dean PR, Tu YL, Xue D. An information system for one-of-a-kind production. Int. J. Prod. Res., 2009; 47(4): 1071-1087.

[5] Dean PR, Xue D, Tu YL. Prediction of manufacturing resource requirements from customer demands in mass-customization production, Int. J. Prod. Res., 2009; 47(5): 1245-1268.

[6] Brucker P, Drexl A, Mohring R, Neumann K, Pesch E. Resource-constrained project scheduling: notation, classification, models, and methods. Eur. J. Oper. Res., 1999; 112(1): 3-41.

[7] Herroelen W, De Reyck B, Demeulemeester E. Resource-constrained project scheduling: a survey of recent developments. Computers & Oper. Res., 1998; 25(4): 279-302.

[8] Goyal SK, Mehta K, Kodali R, Deshmukh SG. Simulation for analysis of scheduling rules for a flexible manufacturing system. Integr. Manuf. Syst., 1995; 6(5): 21-26.

[9] Park SC, Raman N, Shaw MJ. Adaptive scheduling in dynamic flexible manufacturing systems: a dynamic rule selection approach. IEEE Trans. Robot. Autom., 1997; 13(4): 486-502.

[10] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. Eur. J. Oper. Res., 2005; 165(2): 479-494.

[11] King JR, Spachis AS. Heuristics for flow-shop scheduling. Int. J. Prod. Res., 1980; 18(3): 345-357.

[12] Ovacik IM, Uzsoy R. Decomposition Methods for Complex Factory Scheduling Problems. Boston/Dordrecht/London: Kluwer Academic Publishers, 1997.

[13] Tu YL. A framework for production planning and control in a virtual OKP company," Trans. North American Manufacturing Research Institution of SME, 1996; 24: 121-126.

[14] Tu YL. Automatic scheduling and control of a ship welding assembly line. Computers in Industry, 1996; 29(3): 169-177.

[15] Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for the n-job, m-machine scheduling problem. Manage. Sci., 1970; 16(10): 630-637.

[16] Thornton HW, Hunsucker JL. A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. Eur. J. Oper. Res., 2004; 152(1): 96-114.

[17] Pinedo M. Scheduling Theory, Algorithms, and Systems. New Jersey: Prentice Hall, 2002.

[18] Gupta JND, Stafford Jr. EF. Flowshop research after five decades. Eur. J. Oper. Res., 2006; 169(3): 699-711.

[19] Framinan JM, Gupta JND, Leisten R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. J. Oper. Res. Soc., 2004; 55(12): 1243-1255.

[20] Nawaz M, Enscore Jr. EE, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA - Int. J. Manage. Sci., 1983; 11(1): 91-95.

[21] Linn R, Zhang W. Hybrid flow shop scheduling: a survey. Comp. & Ind. Eng., 1999; 37(1): 57-61.

[22] Wang H. Flexible flow shop scheduling: optimum, heuristic and artificial intelligence solutions. Expert Syst., 2005; 22(2): 78-85.

[23] Botta-Genoulaz V. Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. Int. J. Prod. Econ., 2000; 64(1): 101-111.

[24] Li W, Luo X, Tu YL, Xue D. Adaptive production scheduling for one-of-a-kind production with mass customization. Trans. North American Manufacturing Research Institution of SME, 2007; 35: 41-48.

[25] Li W. Adaptive Production Scheduling and Control in One-of-a-Kind Production. Thesis (M.Sc.). University of Calgary, Canada, 2006.

[26] Taillard E. Benchmarks for basic scheduling problems. Eur. J. Oper. Res., 1993; 64(2): 278-285.

[27] Vergara HA, Kim DS. A new method for the placement of buffers in serial production lines. Int. J. Prod. Res., 2009; 47(16): 4437-4456.

Table

Table 1
Characteristics of heuristics H_1 and H_2 for disturbances $D_1$and D_2

Heuristics \ Disturbance	D_1	D_2	Computation time	
			D_1	D_2
H_1	Yes	No	T_{11}	∞
H_2	Yes	Yes	T_{21}	T_{22}

Table 2
Average deviation from Taillard's benchmarks (%)

Scale	Instance	CDS	NIS	SS	SSnoLVR
20*5	10	9.05	7.41	9.14	7.80
20*10	10	13.48	9.46	10.18	13.13
20*20	10	11.07	7.30	10.64	14.02
50*5	10	7.15	4.96	3.60	3.38
50*10	10	14.46	11.57	9.67	9.24
50*20	10	18.13	14.50	16.15	16.12
100*5	10	5.25	4.70	1.60	1.75
100*10	10	9.51	8.27	6.71	6.05
100*20	10	16.45	13.50	11.83	15.71
200*10	10	7.55	6.61	3.09	2.48
200*20	10	13.75	11.33	9.10	11.31
500*20	10	9.56	8.44	5.63	4.60
Total Average		11.28	9.01	8.11	8.80

Table 3
Improvement over NIS heuristic for $F_m/nwt/C_{max}$ problems (%)

Scale	Instance	CDS	SS
20*5	10	-0.32	2.01
20*10	10	-2.59	-2.86
20*20	10	-3.50	-2.71
50*5	10	0.29	8.29
50*10	10	-1.29	0.49
50*20	10	-2.42	-1.67
100*5	10	-0.27	9.20
100*10	10	-0.61	3.78
100*20	10	-1.00	-0.02
200*10	10	-0.22	5.59
200*20	10	-0.41	1.69
500*20	10	-0.10	3.46
Total Average		-1.04	2.27

Table 4
Improvement over NIS heuristic for *FFs/nwt/*C_{max} problems (%)

Scale	Instance	CDS	SS
20*5	10	-1.71	-2.66
20*10	10	-2.72	-2.02
20*20	10	-3.06	-2.88
50*5	10	-0.77	3.34
50*10	10	-1.50	-2.18
50*20	10	-3.48	-2.04
100*5	10	0.21	7.15
100*10	10	-0.55	0.60
100*20	10	-1.75	-1.13
200*10	10	-0.15	3.54
200*20	10	-0.50	0.97
500*20	10	0.01	2.00
Total Average		-1.33	0.39

Table 5
Improvement over CDS heuristic for *FFs/FCFS/*C_{max} problems (%)

Scale	Instance	min (C_{max})	max (*Util*)
20*5	10	-2.39	7.33
20*10	10	0.27	5.66
20*20	10	-2.65	-0.02
50*5	10	2.87	4.90
50*10	10	2.47	6.17
50*20	10	0.08	1.45
100*5	10	2.42	3.47
100*10	10	1.34	4.69
100*20	10	1.54	2.43
200*10	10	3.14	3.96
200*20	10	2.03	4.41
500*20	10	2.79	3.14
Total Average		1.16	3.96

Table 6
Improvement adaptive control over no adaptive control
Where $SPACE_{is} = p_{is+1}$

Scale	Instance	min (C_{max})
20*5	10	2.46
20*10	10	1.81
20*20	10	3.01
50*5	10	0.88
50*10	10	2.17
50*20	10	-2.80
100*5	10	0.39
100*10	10	0.29
100*20	10	-4.18
200*10	10	-0.41
200*20	10	-1.25
500*20	10	-1.21
Total Average		0.10

Table 7
Improvement adaptive control over no adaptive control
Where $SPACE_{is} = c_{is+1} - A_s$

Scale	Instance	min (C_{max})
20*5	10	7.75
20*10	10	6.62
20*20	10	-8.49
50*5	10	1.23
50*10	10	3.10
50*20	10	2.99
100*5	10	0.22
100*10	10	3.26
100*20	10	4.50
200*10	10	0.55
200*20	10	1.64
500*20	10	0.86
Total Average		2.02

Table 8
Improvement adaptive control over no adaptive control with integration of SS heuristic with a weighted factor
Where $SPACE_{is} = p_{is+1}$

Scale	Instance	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
20*5	10	-0.59	0.75	2.17	3.12	1.97	2.46	-2.31	-7.08	-7.68	-9.25	-2.30
20*10	10	0.67	0.77	0.56	5.01	3.68	1.81	-2.86	-4.62	-2.11	-1.59	-4.27
20*20	10	7.50	10.00	8.69	6.74	6.15	3.01	0.00	-9.46	-9.12	-8.99	-9.79
50*5	10	0.22	1.11	1.17	0.97	1.55	0.88	0.15	-1.66	-1.33	-1.24	-0.64
50*10	10	5.50	4.97	4.06	2.80	2.75	2.17	-4.35	-7.10	-6.20	-8.76	-8.89
50*20	10	1.65	3.67	1.94	2.90	-5.41	-2.80	-5.53	-5.22	-7.50	-0.22	-1.07
100*5	10	0.43	0.88	0.36	0.49	0.58	0.39	0.17	-0.98	-1.09	-1.16	-2.24
100*10	10	3.05	3.02	2.20	2.42	1.05	0.29	-2.35	-1.97	-2.76	-1.42	-2.50
100*20	10	5.94	5.24	5.79	4.76	-0.39	-4.18	-5.26	-5.76	-6.61	-5.73	-7.49
200*10	10	0.47	0.47	0.05	0.27	0.15	-0.41	-0.75	-1.26	-1.10	-1.58	-1.15
200*20	10	1.53	2.24	2.39	0.82	-0.15	-1.25	-1.44	-1.48	-2.34	-2.36	-1.98
500*20	10	0.17	0.18	0.16	-0.02	-0.64	-1.21	-1.63	-1.69	-1.66	-1.59	-1.54
Average		2.21	2.77	2.46	2.52	0.94	0.10	-2.18	-4.02	-4.96	-5.33	-6.15

Table 9
An industrial case study

	Gienow	SS	Improvement			Gienow	SS	Improvement
1	1,795	1,711	84		16	1,489	1,489	0
2	1,458	1,444	14		17	1,477	1,477	0
3	1,698	1,697	1		18	1,743	1,712	31
4	2,292	2,261	31		19	1,751	1,745	6
5	1,570	1,556	14		20	1,434	1,430	4
6	1,798	1,753	45		21	1,587	1,570	17
7	1,420	1,420	0		22	1,587	1,393	194
8	1,573	1,567	6		23	1,196	1,165	31
9	1,828	1,805	23		24	1,094	1,083	11
10	1,676	1,676	0		25	1,362	1,362	0
11	1,568	1,568	0		26	1,281	1,281	0
12	1,691	1,691	0		27	923	923	0
13	1,465	1,465	0		28	857	851	6
14	1,364	1,353	11		Total	42,300	41,771	529
15	1,323	1,323	0		Percent			1.25%

Table 10
A two-machine flow shop example

	Machine 1	Machine 2
Job 1	5	20
Job 2	20	10
Job 3	10	15
Job 4	15	12