A STATE-SPACE HEURISTIC FOR DAPTIVE PRODUCTION CONTROL IN ONE-OF-A-KIND PRODUCTION

Wei Li
Department of Mechanical and Manufacturing Engineering
University of Calgary
Calgary, Alberta T2N 1N4

Deyi Xue
Department of Mechanical and Manufacturing Engineering
University of Calgary
Calgary, Alberta, Canada T2N 1N4

Barrie R. Nault
Management Information Systems Area, Haskayne School of Business
University of Calgary
Calgary, Alberta, Canada T2N 1N4

Yiliu Tu
Department of Mechanical and Manufacturing Engineering
University of Calgary
Calgary, Alberta, Canada T2N 1N4

KEYWORDS

Adaptive Production Control, Flow Shop, Machine Breakdown, One-of-a-Kind Production (OKP), Operator Absence.

ABSTRACT

One-of-a-kind production (OKP) is a competitive production mode aiming at achieving mass customization. Whereas, dynamics in OKP cause difficulties on production scheduling, which may be manifested as frequent changes of customer orders, differences of production processes, different processing times of products, and so on. For production scheduling in OKP, we propose a state-space (SS) heuristic. Based on case studies on Taillard's benchmarks, SS shows improvement over the best of six heuristics, CDS. SS also shows the suitability especially for adaptive production control to deal with disturbances, such as operator absence/machine breakdown.

INTRODUCTION

Mass customization is one of the most important competitive strategies in the current economy, the objective of which is to maximize customer satisfaction with highly customized products and near mass production efficiency (Blecker and Friedrich, 2006). One-of-a-kind production (OKP) is a form of mass customization where small- and medium-sized enterprises use a product-oriented positioning strategy and customer services of make to order, assembly to order, and/or engineer to order (Wortaman et al., 1997). Compared with mass production, OKP can achieve high customization but its production efficiency is relatively low.

OKP is characterized by changes/dynamics (Tu, 1997), and every product differs from others on matters of colours, shapes, dimensions, functionalities, materials, processes, processing times, and so on. Such differences present substantial difficulties in production scheduling and control, and to manage these difficulties OKP companies use mixed-product production in a flow line continually employing technologies and resources to reduce production cost, since in general, flow shop production has higher production efficiency than job-shop or open-shop production.

It is difficult to adaptively control the flow shop production, because of the following reasons. High computational complexity makes some heuristics inapplicable to real-time adaptive control, especially for those based on neural net-

works, genetic algorithm and tabu search (Li et al., 2006; 2007). Based on some unrealistic assumptions, such as no operator absence that is the same as no machine breakdown, heuristics are inflexible to deal with disturbances in production (Maccarthy and Liu, 1993; McKay et al., 2002), making scheduling systems inflexible (Kouvelis et al., 2005).

Based on the discussion above, we propose a flexible heuristic in this paper, state-space (SS), for flow shop production. Integrated with our formerly proposed production scheduling and control scheme (Li et al., 2007), SS improves the flexibility of a scheduling and control system for OKP.

ADAPTIVE PRODUCTION SCHEDULING AND CONTROL SCHEME

Our formerly proposed scheduling and control scheme is mainly based on feedback control, a simulation model, THOCPN-CS (Li, 2006), and a heuristic.

For one day production, production planning takes place first before production, generating a task resource matrix (see Figure 1) according to static information, which includes operator numbers for each work station *s*, job numbers *N*, processing times of each job in each station (p_{is}), and so on; such matrix is an input to the scheduling heuristic and simulation model. In this planning phase, i.e. a loop with thin lines, a satisfying production schedule will be generated to be sent to the shop floor. Then in a production phase, i.e. a loop with thick lines, when disturbances happen to production, e.g. emergent job insertion or cancellation, operator absence/machine breakdown, the matrix will be updated, i.e. the disturbances are fed back to the control system, and then the heuristic and simulation model will generate an adaptive solution accordingly. In this production phase, the computation time of a heuristic is critical.

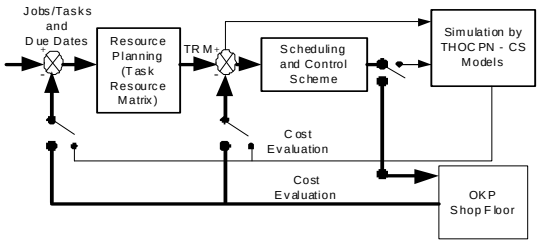

FIGURE 1. AN ADAPTIVE PRODUCTION SCHEDULING AND CONTROL STRUCTUR.

STATE-SPACE HEURISTIC

State-space (SS) heuristic is to min (C_{max}) for *n*-job *s*-station hybrid flow shop (HFS) problems, where there are multiple operators/machines in each station. Because there are multiple operators in a station, to maximize the utilization of a flow line, max (*Util*), is another objective of SS.

The Main Idea of SS

The main idea of SS is as follows. For example, there is a hybrid flow line with 3 work stations and 2 operators in each station (Figure. 2). All operators follow a first available first serve (FAFS) rule, that is, in a station, the operator

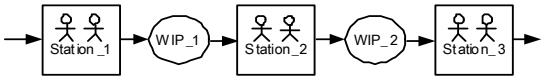

FIGURE 2. A 3-STATION HYBRID FLOW LINE WITH 2 OPERATORS IN EACH STATION.

who finishes a job in hand first should process the next job. Then there is an available time of each station, namely A_s, where A_s = min (a_{sk}), for $k = 1 \ldots OPTR_s$, where a_{sk} is the available time of operator k in station s, and $OPTR_s$ is the number of operators allocated to station s. For an S-station flow line, there are S–1 time differences between station available times. In the example above, there are two available time differences, A_2–A_1, and A_3–A_2. If we regard such a time difference as a space, $SPACE_s = A_{s+1}–A_s$, for $s = 1 \ldots S$-1, then it actually is a time period allowed for station s to finish a job without causing idle in station s+1. If the completion time of job i in station s is longer than the available time of station s+1, then such job causes idle in station s+1, $IDLE_{is} = c_{is} – A_{s+1}$, where c_{is} is the completion time of job i in station s, c_{is} = max (A_s, c_{is-1}) + p_{is}, where p_{is} is the processing time of job i in station s. If the completion time of a job i in station s is earlier than the available time of station s+1, then two cases should be checked. In case one, if the work-in-process (WIP) inventory after a station s, WIP_s, is full, then a delay happens to the operator k who processed job i in station s, $DELAY_{is} = A_{s+1} – c_{is}$. Such a delay means that, after finishing job i, the operator k in station s has to hold it in hand for $DELAY_{is}$ time units until there is a vacancy in WIP_s. Therefore, the available time of operator k in station s is delayed. In case two, if the WIP inventory, WIP_s, is not full, then job i goes into the inventory. In this case, there is no *DELAY* and no *IDLE*.

The main idea of SS is to find a job that fits S–1 spaces, without causing *IDLE* or *DELAY*. After a chosen job i is processed on line, the state of the line, i.e. the available times of stations, is changed, and the space is changed accordingly. It is clear that greater *DELAY* and *IDLE* 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 *DELAY*, *IDLE*, and *SPACE*.

A Lever Concept

From our previous research on traditional flow shop (TFS) problems, in which there is only one operator/machine in each station, we found that the lever concept is suitable to min (C_{max}) (Li et al., 2007), which means that *IDLE* (or *DELAY*) time in an earlier station is worse to min (C_{max}) than in a later station. 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 station flow line as a lever, and $IDLE_{is}$ caused by job i, or $DELAY_{is}$, has a torque effect manifested as $T\_IDLE_{is} = IDLE_{is} \times LVR\_IDLE_s$ or $T\_DELAY_{is} = DELAY_{is} \times LVR\_DELAY_s$, where LVR_IDLE_s or LVR_DELAY_s is the length of arm for *IDLE* or *DELAY*, respectively.

A lever concept for *IDLE* in SS is shown in Figure 3. For an S-station flow line, a job could

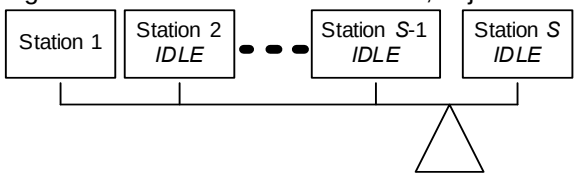

FIGURE 3. A LEVER CONCEPT OF *IDLE* IN SS.

cause at most S–1 times of *IDLE*, since no *IDLE* is caused in station 1 and an *IDLE* takes effect in the next station. Therefore, the fulcrum of a lever for *IDLE* is set between station S–1 and station S, and the length of arm for an *IDLE* caused by station s in station s+1 is $LVR\_IDLE_s = S - s$, for $s = 1 \ldots S\text{-}1$.

A lever concept for *DELAY* in SS is shown in Figure 4. Like the number of possible *IDLE*s,

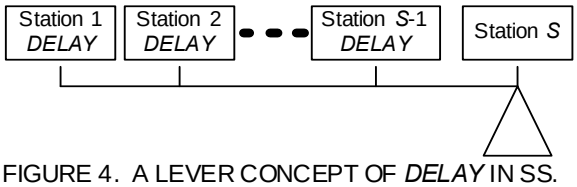

FIGURE 4. A LEVER CONCEPT OF *DELAY* IN SS.

there could be S–1 *DELAY*s, since no *DELAY* is caused in station S. But a *DELAY* takes effect in current station s, whereas *IDLE* is for the next station. Therefore, one unit of *DELAY* should be worse to min (C_{max}) than one unit of *IDLE* in station 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 station S.

There is also a lever concept for *SPACE* in SS, shown in Figure 5. The length of arm for a

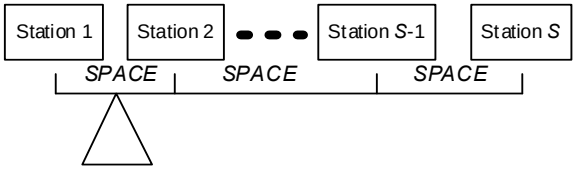

FIGURE 5. A LEVER CONCEPT OF *SPACE* IN SS.

SPACE is $LVR\_SPACE_s = s$, for $s = 1 \ldots S\text{-}1$. The fulcrum of a lever for *SPACE* is set between station 1 and station 2, which means *SPACE* in a later station is better to min (C_{max}) than in an earlier station. Therefore, the torque effect of *SPACE* of a job i is $T\_SPACE_{is} = SPACE_{is} \times LVR\_SPACE_s$.

Generally, there are two kinds of factors affecting the selection of the next job in SS. (1) *SPACE* has a positive effect – the larger the space, the less possible the next job causes *IDLE*; and (2) *IDLE* or *DELAY* has a negative effect – the larger the value, the longer the makespan. A weighting factor, α, is introduced to identify the relationship between these two kinds of factors, using max $[(1 - \alpha) \times T\_SPACE_{is} - \alpha \times (T\_IDLE_{is} + T\_DELAY_{is})]$ to select the next job, where α changes from 0 to 1 with increments of 0.1.

Initial Job Selection

The literature on HFS scheduling is equivocal about whether the longest processing time first rule (LPT) or the shortest processing time first rule (SPT) is better to min (C_{max}) (Wang, 2005). This may be caused by different data in case studies, which were randomly generated by individual research teams. However, because of the complexity of HFS problems, no single rule can be used to consistently generate good solutions for all data sets, especially for OKP where no jobs are the same. Moreover, the main idea of SS is to select successive jobs according to the state space generated by previous jobs in a sequence. Therefore, it is important to select initial jobs to create 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, and the other is the selection scheme. The number of initial jobs is set as min ($OPTR_s$, for s = 1…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 station) is zero since all operators on all stations are available at time zero; if the number is greater than min ($OPTR_s$), then the number of jobs (initial job number – min ($OPTR_s$)) is not selected by the state space concept.

For the selection scheme of initial job selection, five 1×S vectors are introduced, and the initial number of jobs are selected according to $\min\left(\sum_{s=1}^{S}\left(\left|p_{is} - Vector\_v(s)\right|\right), for \quad i = 1...N\right)$, which means the minimum absolute difference between one job's processing times and the vector. Five vectors are *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 station s; *Vector*_5 = [max (p_{is}), i = 1…N] for s = 1…S, i.e. the maximum processing time in each station; *Vector*_2 = *Vector*_3 / 2, and *Vector*_4 = *Vector*_3 + [*Vector*_5 – *Vector*_3] / 2. Therefore, not only *Vector*_1 (like the SPT rule) and *Vector*_5 (like the LPT rule) but also the other three vectors are used for initial job selection. Consequently, there are totally five sequences generated by SS, and the one with the best performance is selected as the final schedule.

SS Programming Logic

The SS heuristic is summarized in the following steps. These steps represent the generic programming logic of SS.

Step 1: Determine the number of operators in each work station, 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 station. (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×S zero vector, Sequence_v. (1g): Set α, the weighting factor for *IDLE* and *DELAY*. This could also be an iteration loop.

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 $\min\left(\sum_{s=1}^{S}\left(\left|p_{is} - Vector\_v(s)\right|\right), for \quad i = 1...N\right)$. Then put selected jobs into a *Sequence_v* and eliminate them from the candidate pool. Calculate the available time of each operator, and then the available time of each station, namely *STATE*, and WIP inventory status, namely *WIP_Status*, which is now a 1×(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 $SPACE_{is}$. To reflect impact of individual jobs on *SPACE*, $SPACE_{is} = c_{is} - A_{s\text{-}1}$.

Step 8: Select job i according to max [(1 – α) × T_SPACE_{is} – α × (T_IDLE_{is} + T_DELAY_{is})], and then put such job number into *Sequence_v* and eliminate it from the candidate pool.

Step 9: Calculate intermediate makespan for all selected jobs, update *WIP_Status*, and update *STATE*.

Step 10: END i. Calculate the utilization of a line. (10a): Calculate utilization of each station first, $Util_s = (\sum_{j=1}^{N} p_{js} \,/\, OPTR_s) \,/\, (c_{Nks} - c_{1k'\text{s-}1})$, c_{1k0} = 0, s = 1…S, in which c_{Nks} is the completion time of the last finished job in station s, $c_{1k'\text{s-}1}$ is the start time of the first job in station s, i.e. the completion time of the first finished job in station s-1, and index k is for operators. (10b): Calculate the average utilization of each station, i.e.

the utilization of a line, *Util* = average ($Util_s$), s = 1…S.

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

Computational Complexity of SS

SS heuristic consists of two parts of calculation. One is to select a job among N jobs, and the other is to calculate intermediate makespan for the selected jobs, that is, to calculate the state space of the flow shop for selected jobs. When comparing two heuristics, every sequence generated by the heuristics should be evaluated based on the objective, e.g. by calculating the makespan. We first examine the computational complexity of the makespan calculation, and then evaluate job selection in SS.

Computational Complexity of Makespan Calculation. An N job and S station 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 stations. The makespan calculation could be carried out along the column dimension. It means that, if the input sequence of N jobs of station 1 is known, then the output sequence of N jobs of station 1, which is also the input sequence of station 2, can be calculated; the output sequence is in an ascending order of completion times of N jobs; and then the output sequence of station 2 can be calculated, and so on, finally the output sequence of station S can be calculated. However, the capacities of WIP inventories are limited, which means the completion times of jobs in station s are constrained by the available times of operators in station s+1. For example, when calculating the output sequence of station s, if a job i's completion time in station s causes the WIP_s inventory overloaded, then a *DELAY* happens to such job i. Such *DELAY* means the job i's completion time is delayed to a later time, and so is the available time of operator k, who processes the job i in station s. Consequently, such a *DELAY* affects the completion times of all jobs successive to job i in station s, and the completion times in the previous station need to be recalculated. In an extreme situation, when a *DELAY* happens in station S-1, the completion times of jobs in all previous stations have to be recalculated. Consequently, it is time consuming to calculate makespan along the column dimension when WIP inventories are constrained. Thus, the makespan calculation should be along the row dimension, i.e. calculated by jobs and not by stations. Therefore, for an N job S station HFS problem, the makespan should be calculated by the following way, first, for one job, then two jobs, and so on, until for N jobs, which means there are totally $1S + 2S + \ldots + NS = \left(\sum_{i=1}^{N} i\right) \times S$ operations. Generally, N is much larger than S and $OPTR_s$, therefore, the computational complexity of the makespan calculation is $O(N^2S)$.

Computational Complexity of Job Selection in SS. The main scheme of job selection in SS is that, first, select one out of N jobs, then one out of N-1 jobs, and so on, until there is only one job left. So there are totally $NS + (N\text{-}1)S + \ldots + S = \left(\sum_{i=N}^{1} i\right) \times S$ operations. Thus the computational complexity of job selection in SS is $O(N^2S)$.

Totally, the computational complexity of SS is $O(N^2S)$.

Summary

Using the state space concept, SS will generate five sequences, and the one with the best performance on objectives is selected as the final schedule. The total computational complexity of SS is $O(N^2S)$.

However, for adaptive control, if the state of production line is known, which means if the state is fed back to the center control system, it takes only NS operations for SS to select next job to deal with disturbances.

CASE STUDIES

There are two types of case studies. The first one is based on Taillard's benchmarks for TFS problems and HFS problems, comparing SS with CDS heuristic (Campbell et al., 1970). The second one is for adaptive control on operator absence.

CDS is an extension of Johnson's algorithm (Johnson, 1954) to min (C_{max}) for N job S station TFS problems. There are totally S-1 sequences generated by CDS, and the one with the best performance on min (C_{max}) is selected as the

final solution. The computational complexity of CDS is $O(NS^2 + SN\log N)$. In 2000, Botta-Genoulaz cleverly converted a HFS problem to a TFS problem and the objective is to min (C_{max}). Botta-Genoulaz concluded that CDS is the best of six heuristics to min (C_{max}) for HFS problems.

TFS Problems

Case study of TFS problems is based on the deviation (DEV) from the best known upper bounds (UB) of Taillard's benchmarks, where DEV = (Makespan of SS or CDS – UB) / UB in percentage.

Table 1 shows the deviations of CDS and SS. There are 12 scales in Taillard's benchmarks, ranging from 20 jobs 5 stations (20*5) to 500 jobs 20 stations (500*20), and there are 10 instances in each scale. The first column in Table 1 is for instance numbers, the second column for scales, the DEV of CDS is in column 3, and columns 4 to 14 represent the performance of α, changing from 0.0 to 1.0. The best performance of one α is highlighted in a bold number.

For TFS problems, total average deviation of CDS from Taillard's benchmarks is 11.28%, but SS's is 9.74% with $\alpha = 0.5$.

HFS Problems

Case study of HFS problems is also based on Taillard's benchmarks. For SS, we set expected throughput rate $r = 31$, since the average processing time of a station in all 120 instances changes from 30.75 to 64.40, therefore the number of operators in each station varies from 1 to 3 by $OPTR_s$ = Roundup (APT_s / r); and we set $WIP_s = 5$ for all WIP inventories and for all instances, which could be set differently.

Table 2 shows the improvement of SS's performance over CDS's based on Taillard's benchmarks. The overall average improvement that SS made over CDS on min (C_{max}) is 1.48% with $\alpha = 0.7$, on max (*Util*) is 4.37% with $\alpha = 0.5$.

TABLE 2. IMPROVEMENT OVER CDS.

	Max Improvement		Relative α	
	C_{max}	*Util*	C_{max}	*Util*
Ta001-010	-1.32%	7.73%	0.6	0.4
Ta011-020	0.46%	6.39%	0.7	0.2
Ta021-030	-1.55%	5.90%	0.7	0.1
Ta031-040	3.28%	4.50%	0.7	0.6
Ta041-050	2.11%	6.11%	0.6	0.7
Ta051-060	1.98%	6.29%	0.6	0.2
Ta061-070	2.79%	3.42%	0.6	0.5
Ta071-080	2.06%	4.98%	0.8	0.5
Ta081-090	1.70%	4.07%	0.7	0.3
Ta091-100	3.70%	3.88%	0.7	0.6
Ta101-110	2.48%	4.53%	0.8	0.7
Ta111-120	4.30%	3.59%	0.8	0.8
Average	1.48%	4.37%	0.7	0.5

Computation Time

Generally in manufacturing, the number of jobs N is much larger than the number of stations S, CDS takes less computation time than SS to generate sequences: the computational complexity of CDS is $O(NS^2 + SN\log N)$ and that of SS is $O(N^2S)$. However, from an application perspective, each of S–1 sequences generated by CDS should be evaluated. Thus, CDS and SS have the same computational complexity, $O(N^2S)$ to determine the final schedule. There-

TABLE 1. DEVIATION FROM TAILLARD'S BENCHMAKRS FOR TFS PROBLEMS (IN PECENTAGE).

	Scale	CDS	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Ta001-010	20*5	9.05	10.95	9.26	8.58	8.13	8.04	**7.75**	8.62	8.53	9.62	8.56	9.19
Ta011-020	20*10	13.48	14.56	13.88	13.55	12.38	**12.31**	12.34	13.20	13.87	14.25	14.09	14.99
Ta021-030	20*20	11.07	12.50	11.07	**10.41**	11.35	12.21	12.24	13.22	14.30	14.5	14.04	14.59
Ta031-040	50*5	7.15	5.51	5.03	5.04	5.41	5.25	5.51	5.31	4.98	4.88	**4.50**	5.98
Ta041-050	50*10	14.46	15.37	14.28	15.21	14.69	12.74	11.94	12.41	11.93	12.06	**11.20**	11.48
Ta051-060	50*20	18.13	20.64	19.31	18.6	18.22	18.82	16.76	16.86	16.68	**15.17**	15.76	17.01
Ta061-070	100*5	5.25	3.17	2.88	2.85	2.71	**2.48**	**2.48**	2.54	2.76	**2.48**	2.52	3.15
Ta071-080	100*10	9.51	10.46	9.90	9.43	8.07	7.89	7.35	7.60	**6.75**	6.98	6.91	7.81
Ta081-090	100*20	16.45	18.32	17.65	17.17	17.00	**15.76**	15.97	16.94	17.05	16.78	17.24	16.3
Ta091-100	200*10	7.55	7.65	7.11	6.25	5.64	5.35	4.55	4.17	3.97	**3.79**	3.95	4.26
Ta101-110	200*20	13.75	16.74	16.36	15.52	13.65	12.69	12.19	12.59	12.05	12.38	**11.71**	12.63
Ta111-120	500*20	9.56	11.27	11.37	10.83	9.98	8.16	7.74	7.16	**6.83**	6.89	7.25	8.10
Total Average		11.28	12.26	11.51	11.12	10.60	10.14	**9.74**	10.05	9.98	9.98	9.81	10.46

fore, for HFS problems, when S is large, CDS takes longer computation time than SS. Table 3 shows this conclusively.

TABLE 3. COMPUTATION TIMES OF CDS AND SS ON TAILLARD'S BENCHMARKS (IN SECONDS).

	Scale	CDS	SS
Ta001-010	20*5	0.09	0.32
Ta011-020	20*10	0.4	0.57
Ta021-030	20*20	1.98	1.15
Ta031-040	50*5	0.31	0.99
Ta041-050	50*10	1.63	2.02
Ta051-060	50*20	8.51	3.75
Ta061-070	100*5	1.18	2.46
Ta071-080	100*10	6.02	6.84
Ta081-090	100*20	29.39	14.71
Ta091-100	200*10	23.85	26.44
Ta101-110	200*20	112.47	55.44
Ta111-120	500*20	851.37	396.15

In Table 7, the computation times of CDS represent the time for the makespan calculation only, not both for sequence generation and makespan calculation, because when the number of jobs N is large, the computation time for CDS to generate sequences is negligible, compared with the time to calculate the makespan.

We conclude that SS heuristic is suitable for real-time production control: if the real state of production line is fed back to the center control system, then the computational complexity for SS to select the next successive job is only $O(NS)$, whereas for CDS it remains $O(N^2S)$.

All SS and CDS heuristics are programmed in MatLab, and run on a computer with a Pentium 4, 2.40 GHz CPU and 512 MB RAM.

A Case Study on Operator Absence

Operator absence or machine breakdown happens often in manufacturing industries. For a case study of it, we assume that, for each instance of Taillard's benchmarks, the production is carried out according to the original schedule determined in the planning phase, but when $N/2$ jobs are finished, one operator is absent in station 3, 6, or 11, for S = 5, 10, and 20 respectively.

If the original schedule is not to be changed for such operator absence, we record its completion time as *Original*. If we adaptively change the schedule for the rest $N/2$ jobs, we record the relative completion time as *Adaptive*. The improvement of adaptive control over no control equals to (*Original* – *Adaptive*) / *Original* in percentage. The results are listed in Table 4.

In Table 4, the third column, CMBN, means combination, choosing the best performance of 11 factors; columns 4 to 14 represent performance of 11 factors respectively. The number in each line is the average improvements for 10 instances in a scale, and the best performance of a factor is highlighted in a bold number. The number in the last line is the total average improvement for all 120 instances.

It is obvious that the production should be adaptively adjusted when operator absence happens to the shop floor. The overall improvement is 1.05% with α = 0.5.

TABLE 4. IMPROVEMENT OVER NO ADAPTIVE CONTROL FOR OPERATOR ABSENCE (IN PECENTAGE).

	Scale	CMBN	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Ta001-010	20*5	4.73	2.35	2.90	1.88	2.34	3.07	***3.54***	3.41	1.72	1.15	-0.07	-0.49
Ta011-020	20*10	4.46	2.10	2.03	***2.52***	1.90	1.86	2.20	0.62	-1.04	0.03	-4.20	-3.98
Ta021-030	20*20	1.27	-0.48	-0.59	-0.33	-0.56	***-0.30***	-1.60	-1.46	-1.17	-1.40	-1.99	-3.75
Ta031-040	50*5	1.24	0.48	0.40	0.12	0.27	0.55	***0.57***	0.50	0.25	0.25	-1.31	-1.92
Ta041-050	50*10	3.33	***1.78***	1.03	1.49	1.59	1.07	1.40	0.51	-0.72	-0.71	-2.38	-3.55
Ta051-060	50*20	2.64	0.14	-0.17	-0.86	0.48	0.39	***1.18***	0.82	0.87	-0.69	-1.68	-2.11
Ta061-070	100*5	0.57	-0.26	0.00	-0.05	-0.07	-0.07	0.11	***0.42***	0.18	-0.25	-0.45	-1.64
Ta071-080	100*10	2.62	1.66	1.64	1.24	1.29	1.50	1.67	***2.15***	1.43	0.55	-0.70	-1.15
Ta081-090	100*20	2.52	1.14	1.19	1.44	1.77	1.63	***1.97***	1.09	0.60	-0.49	-2.18	-2.65
Ta091-100	200*10	0.80	0.36	0.45	0.39	***0.52***	0.44	0.28	0.18	0.04	-0.08	-0.70	-1.05
Ta101-110	200*20	1.41	0.74	0.62	0.75	0.56	0.84	***0.88***	0.30	-0.09	-0.84	-0.86	-1.00
Ta111-120	500*20	0.80	0.36	0.35	0.47	0.46	***0.57***	0.47	0.42	0.29	0.04	-0.33	-0.63
Total Average		2.20	0.86	0.82	0.76	0.88	0.96	***1.05***	0.75	0.20	-0.20	-1.40	-1.99

CONCLUSIONS

OKP is a typical production mode providing high customized products. However, production in OKP is under dynamic disturbances, because of the nature of OKP. To improve the production efficiency of OKP, we proposed a production scheduling scheme and a state-space (SS) heuristic to achieve adaptive production scheduling and control. SS heuristic is better than CDS on min (C_{max}) for both traditional flow shop problems and hybrid flow shop problems, in which CDS is the best of six heuristics (Botta-Genoulaz, 2000), and SS is flexible and efficient to deal with operator absence/machine breakdown and job insertion/cancellation.

Our future work would be to further improve SS heuristic for other kinds of disturbances, e.g. variance on processing times, and to reduce the sensitivity of SS to the weighting factor α.

ACKNOWLEDGEMENT

This research project has been funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) through its strategic project program.

REFERENCES

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

Botta-Genoulaz, V. (2000). "Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness." *International Journal of Production Economics*, Vol. 64, no. 1-3, pp. 101-111.

Campbell, H.G., R. A. Dudek, and M. L. Smith (1970). "A heuristic algorithm for the n-job, m-machine scheduling problem." *Management Science*, Vol. 16, pp. 630-637.

Johnson, D. S. (1954). "Optimal two- and three-station production schedules with set-up times included." *Naval Research Logistics Quarterly*, Vol. 1, pp. 61-68.

Kouvelis, P., C. Chambers, and D.Z. Yu (2005). "Manufacturing operations manuscripts published in the first 52 issues of POM: review, trends, and opportunities." *Production and Operations Management*, Vol. 14, no. 4, pp. 450-467.

Li, W. (2006). *Adaptive production scheduling and control in one-of-a-kind production*. M.Sc. Thesis, Department of Mechanical and Manufacturing Engineering, University of Calgary, Calgary, Alberta, Canada.

Li, W., X.G. Luo, Y.L. Tu, and D. Xue (2007). "Adaptive production scheduling for one-of-a-kind production with mass customization." *Transactions of NAMRI/SME,* Vol. 35*,* pp. 41-48.

Li, W., Y.L. Tu, and D. Xue (2006). "Adaptive production scheduling and control for one-of-a-kind production shop floor." *Transactions of NAMRI/SME,* Vol. 34*,* pp. 159-166.

Linn, R., and W. Zhang (1999). "Hybrid flow shop scheduling: a survey." *Computers & Industrial Engineering*, Vol. 37, no. 1-2, pp. 57-61.

Maccarthy, B.L., and J. Liu (1993). "Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling." *International Journal of Production Research*, Vol. 31, no. 1, pp. 59-79.

McKay, K., M. Pinedo, and S. Webster (2002). "Practice-focused research issues for scheduling systems." *Production and Operations Management*, Vol. 11, no. 2, pp. 249-258.

Ruiz, R., and C. Maroto (2005). "A comprehensive review and evaluation of permutation flowshop heuristics." *European Journal of Operational Research*, Vol. 165, pp. 479-494.

Taillard, E. (1993). "Benchmarks for basic scheduling problems." *European Journal of Operational Research*, Vol. 64, no. 2, pp. 278-285.

Tu, Y.L. (1997). "Production planning and control in a virtual OKP company." *Computers in Industry*, Vol. 34, pp. 271-283.

Wang, H. 2005. Flexible flow shop scheduling: optimum, heuristic and artificial intelligence solutions. *Expert Systems*, Vol. 22, no. 2, pp. 78-85.

Wortmann, J.C., D.R. Muntslag, and P.J.M. Timmermans (Eds.) (1997). *Customer-driven Manufacturing*. Chapman & Hall, London.