Simultaneous Scheduling Strategy: A Novel Method for Flexible Job Shop Scheduling Problem

Bo Liu[1], Siqi Qiu[1,*] and Ming Li[1]
[1] Department of Industrial Engineering and Management, Shanghai Jiao Tong University, Shanghai 200240, China
lbws888@sjtu.edu.cn, siqiqiu@sjtu.edu.cn, nono-MAA@sjtu.edu.cn

***Abstract*—This paper discussed the contradictory between wait time for computation and solution quality in solving flexible job shop scheduling problems. In order to reconcile this contradictory, a novel scheduling strategy called Simultaneous Scheduling is proposed. At first, a fairly good solution is obtained in a short time and the solution is set as the temporary processing plan so that the machines can start to work as soon as possible. Then while the machines are running, the temporary plan is being improved with evolutionary algorithms continuously. Experiments on both static and dynamic scheduling problems are performed. The results show that simultaneous scheduling is effective and efficient.**

***Index Terms*—Flexible job shop scheduling, Scheduling strategy, Evolutionary algorithms.**

I. Introduction

Evolutionary algorithms (EAs) are currently among the most commonly used methods for Flexible Job Shop Scheduling Problem (FJSP) [1], and are also among the most popular methods for Dynamic Flexible Job Shop Scheduling Problem (DFJSP) [2]. EAs like Genetic Algorithm (GA) [3] and Variable Neighborhood Search (VNS) [4] can perform well in solving FJSP, but the cost of time for computing will expand rapidly with the scale of the problem. In FJSP, the increase in the number of the machines or tasks will cause the time required for the convergence of EA to swell. If the objective of scheduling is to save time, excessive cost of time will make the scheduling computing meaningless, especially for job shops with short processing time for each operation. As for DFJSP, the time-consuming problem of EAs is even more prominent, because each rescheduling takes long time to compute. Bierwirth et al. [5,6] proposed a population initialization method that can accelerate GA convergence in DFJSP. This method generates the initial population of GA in rescheduling based on the population of the last generation of the previous scheduling or rescheduling, which is proved to be able to improve the convergence speed and the solution performance.

Besides, there are also some scheduling methods that cost less time than EAs. Dispatch rules such as Shortest Imminent Processing Time (SPT) and First In First Out (FIFO) are widely used in DFJSP [7], which generate schedules based on simple priority rules. Due to their simplicity of strategies, the cost of computing time can be considered as zero [8]. Ziaee et al. [9] proposed a simple heuristic algorithm to solve FJSP which can obtain scheduling solutions whose performances are not much worse than those solutions of GA or VNS while the computing time is several orders of magnitude shorter than GA and VNS. The advantage of using those simple heuristic methods is that they can obtain a fairly good scheduling scheme as soon as possible. However, the gap between the performance of simple heuristic algorithm and EA methods is still significant.

Hence, the contradiction between the time consumption of the scheduling computing and the solution performance is still an important issue which need to be solved. To reconcile this contradiction, this paper proposes a novel scheduling strategy called Simultaneous Scheduling (SS), which focuses on paralleling the runtime of scheduling computing with the runtime of machines processing. In SS, an initial solution is obtained in a short time with simple heuristic algorithms and the solution is set as the temporary scheduling scheme. Then, machines start to process according to the temporary scheme. While the machines are running, the scheme is continuously being improved with EA. This method takes almost no time for computing before the machines start to process, and can be applied to both static FJSP and DFJSP. For DFJSP, this paper mainly concentrates on the dynamic factor of non-deterministic job release times.

The core idea of SS is to utilize the duration of processing to improve the schedule while the machines are running. In most other strategies to solve FJSP and DFJSP [1,2], every scheduling or rescheduling computing is completed at a time before the scheduling scheme is in effect. The disadvantage of these strategies is that the time during processing is wasted and machines often need to wait for the scheduling computing.

The remainder of this paper is organized as follows. Section II formulates the FJSP and DFJSP. Section III gives a detailed introduction of SS as well as design of two algorithms based on SS: GA-based Simultaneous Scheduling (GA-SS) and VNS-based Simultaneous Scheduling (VNS-SS). Experimental studies are presented in Section IV. Section V describes

B. Liu, S. Qiu, and M. Li are with the the School of Mechanical Engineering and Department of Industrial Engineering & Management, Shanghai Jiao Tong University, Shanghai, 200240 China.
Corresponding Address: siqiqiu@sjtu.edu.cn. (Siqi Qiu)

the conclusions and discusses about the applications of SS and future works.

II. Problem Formulation

The SS proposed in this paper is oriented to two types of problems: static FJSP and DFJSP. The static FJSP are deterministic while DFJSP are non-deterministic [6]. In other words, the difference between them is that every condition is assumed to be determined in advance in static FJSP while conditions are changing during processing in DJFSP. The dynamic factors of DFJSP include unexpected machine breakdown, uncertain processing time, stochastic job releases and etc. In this paper, we mainly consider jobs released at the different points of time as the dynamic factor.

A. The static FJSP model

In flexible job shop with the scale of $m \times n$, there are a set of machines $M_1, M_2, ..., M_m$ and a set of jobs $J_1, J_2, ..., J_n$. Each job consists of a sequence of operations $O_{i,1}, O_{i,2}, ..., O_{i,n_i}$. Each operation can be processed on anyone of a subset of machines and the average size of the subset indicates the flexibility of the FJSP. The assumptions and constraints of FJSP in this paper are as follows [9].

- One machine can only process one operation at a time.
- One job can only be processed on one machine at a time.
- Any operation cannot be processed until all the preceding operations of this job are finished.
- Machines have no setup time and the move time between operations are not considered.

Denote the processing time of $O_{i,j}$ on M_k as $T_{i,j,k}$, and the completion time of job J_i is denoted as C_i. The makespan refers to the final completion time of all jobs, which is denoted as C_{max}. The makespan is a common objective of FJSP [1]. C_i and C_{max} are defined by Eq.1 and Eq.2.

$$C_i = \sum_{j=1}^{n_i} T_{i,j,k} \tag{1}$$

$$C_{max} = max\{C_i\} \tag{2}$$

These sets M_k, J_i, $O_{i,j}$ and $T_{i,j,k}$ are determined and known in advance in static FJSP.

B. The DFJSP model

The assumptions and constraints of DFJSP are the same as FJSP, except for that every job J_i has an earliest start time called release time R_i. Any operation of J_i cannot be processed before R_i. In DFJSP, the information of those jobs with R_i later than the present time is non-deterministic and cannot be the used for scheduling. The release time set $\{R_i\}$ is also non-deterministic and the scheduler does not know when a new job will be released. The flow time F_i of job J_i refers to the time duration from the release time to the completion time of J_i. Mean flow time $\overline{F}$ is a common indicator in the literature on DFJSP [10]. F_i and $\overline{F}$ are defined by Eq.3 and Eq.4.

$$F_i = C_i - R_i \tag{3}$$

$$\overline{F} = \sum_{i=1}^{n} F_i / n \tag{4}$$

C. Wait Time at Scheduling Computing

A scheduling scheme is called 'in effect' if it is directly used to schedule the processing. In DFJSP, when new jobs are released, the previous scheduling scheme becomes non-effective and a new scheme in effect should be obtained by rescheduling. As is discussed in Section I, when new jobs are released, the machines are unable to react instantly because it takes a period of time for the scheduler to obtain an effective scheduling scheme and set it to be in effect. The length of that period of time depends on the algorithm adopted. The longer the period is, the more time machines have to wait for a scheduling scheme in effect.

To measure how long machines need to wait, Wait Time at scheduling computing (WT) is introduced. WT is defined by Eq.5. E_i stands for the point of time when the i-th event happens, which changes the FJSP conditions, e.g. that some new jobs are released at time E_i. In the context of static FJSP, E_i refers to the time when all the jobs are determined and the scheduling computing can start. S_i stands for the point of time when the first new scheduling scheme becomes in effect after the event happens at E_i. WT indicates the total amount of time when machines have no effective scheme to follow and thus can not start to work.

$$WT = \sum_{i=1}^{n} S_i - E_i \tag{5}$$

III. Proposed Strategy

A. Overview of proposed SS

The SS makes full use of the duration of machine processing time to perform time-consuming computing. There is almost no wait time from receiving a task to starting processing. It is worth noting that the EAs in SS has no termination, which is different from mainstream methods for job shop scheduling. The computing goes simultaneously with the job processing, since the first operation starts till the last job is finished. After every iteration of the computing, the temporary solution and the best solution of the new EA iteration are compared and the former is replaced if the latter is better.

In order to achieve SS, scheduling solution should keep updated with the real-time situations in the flexible job shop. As is shown in Fig.1, the SS has six main steps:

1) A fixed number of solutions are generated randomly or based on simple heuristic rules quickly;

2) After the initial solutions are obtained, the best one among these solutions is selected as the Temporary Plan (TP);

3) Machines start to work according to the TP;

4) Optimize TP with EA. Every time a new solution that is better than TP is found, replace TP with the better solution;

5) The length of the solutions are continuously updated during processing. Every time an operation starts, the corresponding part is deleted from the solution vectors; every time a new job is released, new operations are inserted into the solution vectors based on certain rules.

6) When all the jobs are finished, the scheduling computing stops.

B. Encoding Scheme

The solutions are encoded based on the scheme in [11,12] which uses two vectors to present a solution. The first one presents the machine assignment (MA), which assigns a machine available for each operation. The MA vector consist of discrete machine indices, which stand for the assigned machines for all the operations in order of $(O_{1,1}, O_{1,2}, ..., O_{1,n_1}, O_{2,1}, ..., O_{n,n_n})$. The second one presents the operation sequence (OS), which determines priority order of the operations. The OS vector consists of discrete job indices, and each index J_i appears in the vector for n_i times. The first J_i in OS vector stands for $O_{i,1}$, the second J_i stands for $O_{i,2}$, and etc.

Fig.2 shows an example of the encoding scheme. This FJSP has 3 machines and 2 jobs containing 3 operations respectively. The first 3 indices of MA present the corresponding assigned machines for the 3 operations of Job J_1 and the last 3 indices correspond to J_2. Fig.3 is the Gantt chart showing the scheduling result of MA and OS in Fig.2. It is worth noting that OS is not always the real processing sequence of the operations but the priority order of them. For instance, $O_{2,3}$ precedes $O_{1,3}$ in OS, but as is shown in the Gantt chart, $O_{2,3}$ starts after $O_{1,3}$.

C. Algorithm design based on SS

Scheduling techniques using SS can be designed based on any EAs. In order to apply EA to SS, some modification of the EA is necessary.

1) Termination criteria are omitted and the EA keeps running until all the jobs are finished.

2) The output of the EA is the manipulation of TP after each iteration. If the best solution in one iteration is better than TP, the solution becomes the new TP and otherwise TP is kept unchanged.

3) To update the solution vectors when new jobs are released, new operation insertion rules should be determined. The insertion methods may use random insertion or heuristic rule-oriented insertion [8].

4) In the computation of fitness functions such as makespan, the start time of different jobs and machines should be considered. At time T, the start time of machines (which is denoted as S_{M_k}) and of job J_i (which is denoted as S_{J_i}) are defined by Eq.6 and Eq.7 [13]. $C_{i,j}$ stands for the completion time of $O_{i,j}$.

$$S_{M_k} = \begin{cases} T, \ \ M_k \text{ is idle at } T \\ C_{i,j}, \ M_k \text{ is processing } O_{i,j} \text{ at } T \end{cases} \tag{6}$$

$$S_{J_i} = \begin{cases} T, \quad \text{No operation of } J_i \text{ is being processed at } T \\ C_{i,j}, \ O_{i,j}\text{is being processed at } T \end{cases} \tag{7}$$

1) GA-SS: GA-SS is designed based on GA in [3], and the encoding scheme introduced above is adopted. The initial TP is selected from the initial population of GA. In order to minimize WT, the initial population should be generated in a short time. Hence, the population initialization method should be studied. In this paper two initialization approaches for GA-SS are adopted:

(1) Random Approach: The initial population is entirely generated based on the initialization method in [3].

(2) Dispatch Rule-Oriented Approach: $s\%$ of the initial population are generated based on the random approach and the rest $t\%$ are generated by dispatch rules SPT, FIFO and Arrival Time (AT).

GA-SS using the random approach and the dispatch Rule-Oriented approach are denoted as GA-SS(rand) and GA-SS(RO) respectively. After every generation of evolution, the best solution among the population is compared with TP to decide whether TP should be updated.

When new jobs are released, the new operations should be inserted into the solution vectors. The operation insertion phase is described as follows.

(1) Random Insertion: $p\%$ individuals of the population are selected from the population based on binary tournament selection. The new operations are inserted randomly into the OS vectors and random machine assignment is added to the MA vectors of the selected individuals.

(2) Appending Insertion: The best $q\%$ individuals of the population are selected. The new operations are appended to the OS vectors and random machine assignment is added to the MA vectors of the selected individuals.

(3) Dispatch Rule Insertion: The rest $r\%$ are generated based on three dispatch rules: SPT, FIFO and AT.

2) VNS-SS: VNS-SS is designed based on the VNS in [4]. After each iteration, the current solution is compared with TP. As GA-SS does, VNS-SS has two initialization approaches. In initialization, a set of candidate solutions, similar to the initial population in GA-SS, is generated and the best solution in the set is selected as the initial solution of VNS. The selected initial solution is also TP. VNS-SS using the random initialization approach and the dispatch rule-oriented approach are denoted as VNS-SS(rand) and VNS-SS(RO) respectively.

When new operations are to be inserted, a new set of candidate solutions is generated. $p\%$ of the set are generated by randomly inserting the new operations into TP; $q\%$ are generated by appending the new operations to TP; $r\%$ are generated based on SPT, FIFO and AT, similar to the three insertion phases in GA-SS.

IV. Experimental Studies

To test the performance of GA-SS and VNS-SS on static FJSP and DFJSP with different scales of processing time

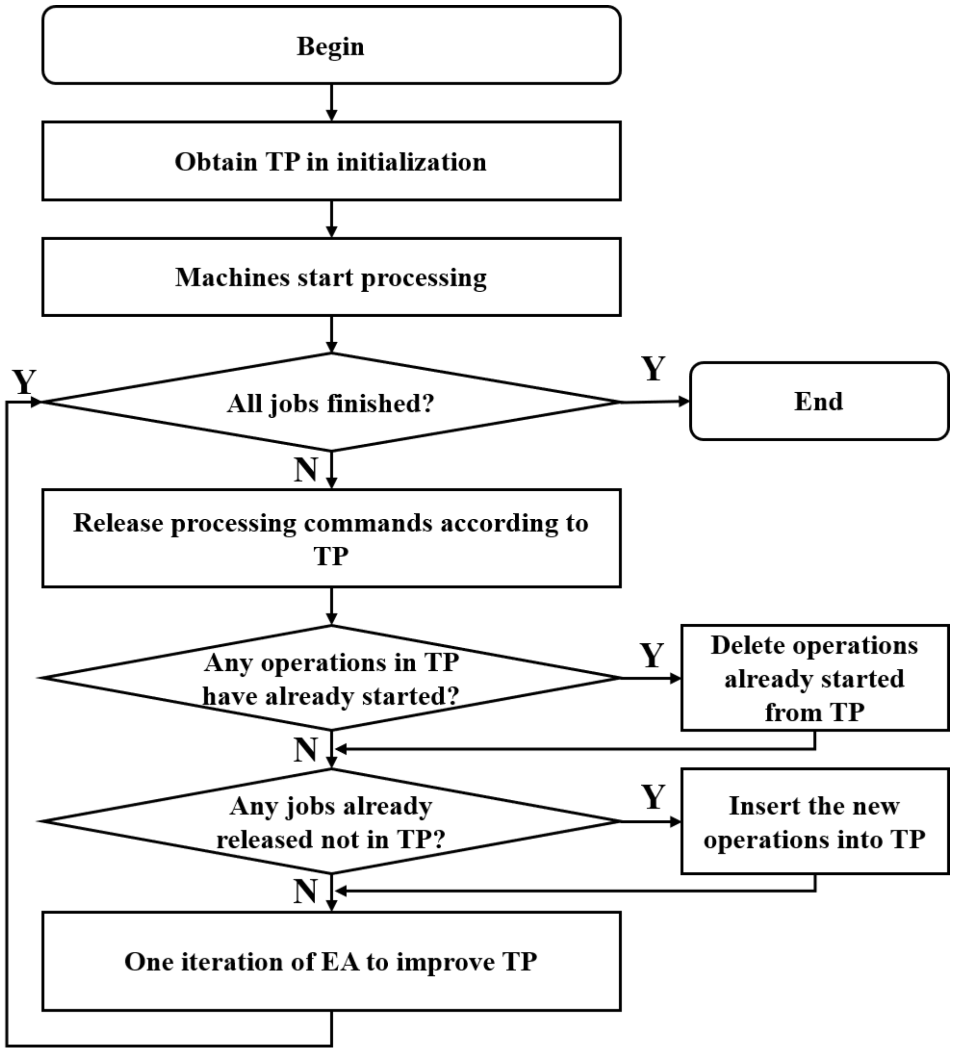
Begin
Obtain TP in initialization
Machines start processing
All jobs finished?
Y
Y
End
N
Release processing commands according to TP
Any operations in TP have already started?
Y
Delete operations already started from TP
N
Any jobs already released not in TP?
Y
Insert the new operations into TP
N
One iteration of EA to improve TP

Fig. 1. The main flowchart of SS.

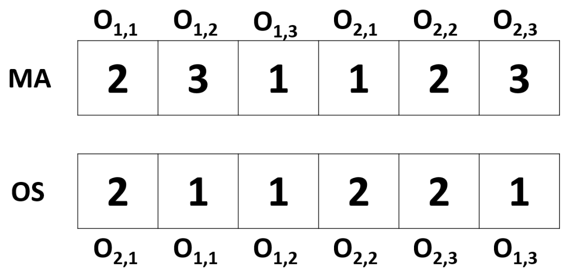

Fig. 2. An example of the encoding scheme.

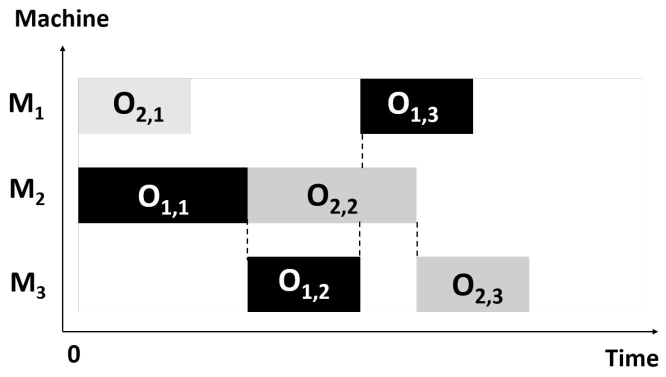

Fig. 3. Gantt chart corresponding to the solution in Fig.2.

$T_{i,j,k}$, two sets of scheduling and processing simulation experiments are performed. Four SS algorithms (GA-SS(RO), GA-SS(rand), VNS-SS(RO) and VNS-SS(rand)) and real-time static and dynamic job shop simulation programs were coded in Python 3 to perform these experiments. The static FJSP experiments are run on a PC with Intel Core i5-5257U CPU/8 GB RAM and the DFJSP experiments are run on another PC with Intel Core i7-8750H CPU/16 GB RAM. The parameters of GA-SS and VNS-SS in the two sets of experiments are shown in Table I.

A. Scale of Processing Time

Because SS utilizes the time duration of processing to compute, the scale of the processing time significantly influence the amount of time the computing could last. Because the complexity of FJSP is only determined by m, n, $\{n_i\}$ and the flexibility, not by the processing times $\{T_{i,j,k}\}$, larger scale of processing time leads to better performance of SS. In the following experiments, problem instances themselves have no unit of time. Integer numbers are used to indicate $T_{i,j,k}$. In order to simulate environments where operations of jobs have different scale of processing times, different units of time including $1sec$, $2sec$, $4sec$ and $8sec$ are adopted.

B. Experiment 1: Static FJSP Simulation

The Brandimarte dataset [14], which contains 10 FJSP problems of various scales, is used in experiment 1. The Brandimarte dataset is commonly used in the literature on FJSP [3,4,9,11]. As is mentioned above, problem instances in Brandimarte dataset have no unit of time in its original setting but the experiments adopt 4 different ones to simulate 4 levels of scales of $T_{i,j,k}$. As is commonly the case in the literature on FJSP[3,4], best makespan and average makespan are the objectives. Each algorithm has been run for 5 times to solve each problem in experiment 1.

Table II presents the information of the problem instances of Brandimarte dataset and the performance of four algorithms in literature used to compare with SS, including GA [3], VNS [4], Simple Heuristics [9] and Dispatch Rule-Oriented Algorithm (ROA) combining SPT and FIFO. The performance data of GA, VNS and Heuristic in Table II are from [3,4,9] and the data of ROA are acquired through experiments. The first two columns show the number of the problem instances and their scales. In the third column, LB denotes the best-known makespan. The fourth to the seventh columns show the results of the four algorithms. BC_{max} and AC_{max} stand for the best makespan and the average makespan respectively. It should be noted that Table II do not present AC_{max} for GA and Heuristic because the corresponding source papers did not report the data. Those data are not important and have no impact on the experiments of this paper.

Table III and Table IV present the average performance of GA-SS and VNS-SS under each scale of time. The results show that VNS-SS outperforms GA-SS. The difference of performance between VNS-SS(RO) and VNS-SS(rand) is insignificant, but GA-SS(RO) outperforms GA-SS(rand) especially under small scales of time. The average performance of VNS-SS is not much worse than VNS and in the majority of the problems the result are close (except for Mk04, Mk06 and Mk10). The average performance of VNS-SS is near to the best performce of Heuristic in [9] and is even better than the best performance of Heuristic in Mk04, Mk05, Mk07 (when Unit = $4s$ or $8s$), Mk08, Mk09 and Mk10 (when Unit = $2s$, $4s$ or $8s$).

For SS, average performance is more important than best performance but [3, 9] only present the best performance. In order to perform a more comprehensive comparison with the algorithms in literature, the best performance of GA-SS and VNS-SS under all scales of time (5 runs for each scale of time and each algorithm) is presented in Table V. The results show that VNS-SS performs better and outperforms Heuristic and ROA. The best performance of VNS-SS is close to GA and VNS in the majority of the problems.

C. Experiment 2: DFJSP Simulation

In experiment 2, the problem instances are generated randomly based on the method in [6]. In real-world job shops, there are situations of different levels of workload and machine utilization rate (which is a common indicator of workload and is denoted as U). U in the range of $0.6-0.9$ can represent the the levels from relaxed workload to excessive workload. Four different problem instances are generated randomly according to the rules following.

(1)The number of machines is $m = 6$.

(2)Each problem include 50 jobs.

(3)The number of operations of each job is uniformly distributed in $\{4, 5, 6\}$.

(4)The number of machines each operation can be processed on is uniformly distributed in $\{1, 2, 3, 4\}$ and the machines are randomly selected.

(5)$T_{i,j,k}$ is uniformly distributed in the range of $[1, 7]$ and can only be integers.

(6)The intervals between job release times is is exponential distributed based on λ.

Thus, the average processing time of a job, which denoted as P, is 20. To adjust the expected U, Eq.8 show how to compute λ.

$$\lambda = P/(m \times U) \tag{8}$$

Because of the dynamic property of DFJSP, only one or few jobs have been released in the beginning. Hence, the initialization approaches do not matter and only two rather than four SS algorithms are tested in experiment 2: GA-SS and VNS-SS. Each algorithm in experiment 2 has been run for 5 times to solve each problem.

Table VI presents the performance of the three algorithms used to compare with SS, including GA [3], VNS [4] and ROA combining SPT, FIFO and AT. All the performance data in Table VI are quired through experiments and the the parameters of GA and VNS are set in the same way with [3,4] except that the size of population and the total number of generations of GA are set 100 and 500 respectively and that the permitted maximum step number with no improving of both GA and VNS is set 50. The first column shows the corresponding U of the problem instances. In the second to the fourth columns show the results of the three algorithms in literature. Table VII show the performance of GA-SS and VNS-SS under different scale of processing times.

The results show that GA-SS outperforms VNS-SS in DFJSP and that GA also performs better than VNS. Under all

TABLE I
PARAMETERS OF GA-SS AND VNS-SS IN EXPERIMENT 1

Algorithms	Parameters	value
GA-SS	The size of the population	100
	Crossover probability, pc	0.8
	Mutation probability, pm	0.1
	Proportion of random initialization of GA-SS(RO), $s\%$	80
	Proportion of RO initialization of GA-SS(RO), $t\%$	20
	Proportion of random insertion phase, $p\%$	20
	Proportion of appending insertion phase, $q\%$	20
	Proportion of dispatch rule insertion phase: $r\%$	60
	Selection phase	Binary tournament
VNS-SS	Size of the candidate solutions	100
	Proportion of random insertion phase, $p\%$	20
	Proportion of appending insertion phase, $q\%$	20
	Proportion of dispatch rule insertion phase: $r\%$	60

TABLE II
PERFORMANCE IN BRANDIMARTE PROBLEMS OF FOUR ALGORITHMS IN LITERATURE

Problem	Scale($n \times m$)	LB	GA [3]	VNS [4]		Heuristic [9]	ROA	
			BC_{max}	BC_{max}	AC_{max}	BC_{max}	BC_{max}	AC_{max}
Mk01	$10*6$	36	40	40	40.0	42	42	43.4
Mk02	$10*6$	24	26	26	26.2	28	37	38.2
Mk03	$15*8$	204	204	204	204	204	204	207
Mk04	$15*8$	48	60	60	60.2	75	73	75.0
Mk05	$15*4$	168	173	173	173.0	179	181	182.6
Mk06	$10*15$	33	63	59	60.0	69	86	91.8
Mk07	$20*5$	133	139	140	140.8	149	191	196.4
Mk08	$20*10$	523	523	523	523.0	555	523	524.6
Mk09	$20*10$	299	311	307	307.8	342	331	341.2
Mk10	$20*15$	165	212	207	208.4	242	269	274.4

TABLE III
AVERAGE PERFORMANCE IN BRANDIMARTE PROBLEMS OF GA-SS

Problem	AC_{max}							
	GA-SS(RO)				GA-SS(rand)			
	Unit=1s	Unit=2s	Unit=4s	Unit=8s	Unit=1s	Unit=2s	Unit=4s	Unit=8s
Mk01	43.0	43.0	42.6	42.4	43.4	42.8	42.2	42
Mk02	33.2	32.4	30.8	31.8	37.0	34.4	34.2	35.2
Mk03	204.0	205.6	204.0	206.8	209.8	210.2	208.8	204.0
Mk04	70.6	69.0	67.8	69.4	70.6	68.6	69.8	71.4
Mk05	179.8	178.6	178.4	177.8	181.0	180.4	178.6	179.6
Mk06	86.0	81.8	79.0	80.2	94.4	90.0	90.6	86.0
Mk07	155.6	152.6	149.8	151.4	167.4	163.8	167.4	168.8
Mk08	523.0	523.0	523.0	523.0	523.2	523.0	523.0	523.0
Mk09	331.0	330.4	330.0	327.2	336.8	337.6	333.4	331.4
Mk10	264.0	265.6	261.2	257.4	276.8	275.4	267.6	264.4

TABLE IV
AVERAGE PERFORMANCE IN BRANDIMARTE PROBLEMS OF VNS-SS

Problem	AC_{max}							
	VNS-SS(RO)				VNS-SS(rand)			
	Unit=1s	Unit=2s	Unit=4s	Unit=8s	Unit=1s	Unit=2s	Unit=4s	Unit=8s
Mk01	42.0	43.4	42.0	42.0	43.6	41.6	42.0	41.8
Mk02	37.2	32.8	30.6	30.4	33.8	30.4	29.2	29.2
Mk03	204.0	204.2	204.0	204.0	204.2	204.0	204.0	205.2
Mk04	69.8	68.0	66.2	67.4	70.2	68.4	69.2	68.8
Mk05	177.8	176.4	174.6	175.0	178.0	174.4	175.2	175.2
Mk06	80.0	72.2	71.2	69.8	82.2	74.6	72.2	71.6
Mk07	155.6	150.2	149.6	146.4	154.0	149.8	148.8	146.2
Mk08	523.0	523.0	523.0	523.0	523.6	524.2	524.8	523.0
Mk09	323.2	319.0	323.0	317.8	322.8	326.2	314.2	316.0
Mk10	247.2	241.8	238.8	235.2	242.4	238.8	236.4	231.8

TABLE V
BEST PERFORMANCE IN BRANDIMARTE PROBLEMS OF VNS-SS (UNIT=1S, 2S, 4S, 8S)

Problem	BC_{max}			
	GA-SS(RO)	GA-SS(rand)	VNS-SS(RO)	VNS-SS(rand)
Mk01	42	41	41	40
Mk02	28	32	29	28
Mk03	204	204	204	204
Mk04	67	65	67	67
Mk05	176	177	173	173
Mk06	77	83	67	70
Mk07	146	159	144	144
Mk08	523	523	523	523
Mk09	319	321	313	311
Mk10	249	253	224	229

TABLE VI
PERFORMANCE IN DFJSP OF THREE ALGORITHMS IN LITERATURE

U	$\overline{F}$		
	GA	VNS	ROA
0.6	14.96	15.00	20.59
0.7	16.20	16.26	20.90
0.8	19.28	19.75	23.60
0.9	23.96	24.92	28.31

TABLE VII
PERFORMANCE IN DFJSP OF GA-SS AND VNS-SS

U	$\overline{F}$							
	GA-SS				VNS-SS			
	Unit=1s	Unit=2s	Unit=4s	Unit=8s	Unit=1s	Unit=2s	Unit=4s	Unit=8s
0.6	15.29	15.12	14.83	14.88	15.91	15.63	15.61	15.41
0.7	16.35	16.22	15.79	15.66	16.72	16.55	16.31	16.40
0.8	18.99	18.83	19.08	19.39	19.95	19.47	19.54	19.23
0.9	24.44	23.86	23.72	23.90	24.72	24.35	23.52	23.44

different U, the performance of GA-SS and VNS-SS is slightly better than GA and VNS respectively and is significantly better than ROA.

D. Results Analysis

The results in the two sets of experiments show that the SS is effective in both static FJSP and DFJSP. In DFJSP, the proposed GA-SS and VNS-SS outperform GA and VNS of the state of art.

The analysis on why GA-SS and VNS-SS outperform the other algorithms is as follows. One reason is that the EA in SS has no termination criteria and the computing goes on for a longer time (although the WT of SS is near zero). The other reason is that the scale and complexity of the whole DFJSP is very large and SS has a natural advantage in solving large scale problems. When using GA, VNS or other EA in common approaches, it is hard to obtain an optimal solution of large scale FJSP. But in SS, with the machine processing going on, the number of the remaining operations to complete is decreasing and the scale of the real-time FJSP is shrinking. As a result, the processing scheme of the remaining operations is becoming easier to optimize. Thus, SS can obtain better scheduling schemes in large scale FJSP. To illustrate this advantage, assume that there are 100 jobs in a FJSP. For GA and VNS, the sequence and machine assignment of all the operations of the 100 jobs are to be scheduled at a time. For GA-SS and VNS-SS, the case is different. In the beginning, SS is solving a FJSP containing 100 jobs; when 50 jobs have been finished, SS is solving a FJSP containing 50 jobs; when 90 jobs have been finished, SS is solving a FJSP containing only 10 jobs. The scale of the problem is becoming smaller and smaller and the optimal scheduling scheme of the remaining operations is becoming easier and easier to obtain.

V. DISCUSSION AND FUTURE WORK

This paper proposes the novel scheduling strategy, in which scheduling schemes are continuously optimized in simultaneity with machine processing. This strategy can be used to minimize wait time for computing so that the machines can start to work as soon as possible and can obtain scheduling solutions with high quality.

Two algorithms GA-SS and VNS-SS are presented and simulation experiments on them are performed. The experimental studies show that GA-SS and VNS-SS significantly outperform simple heuristic algorithms of the state of art and that solution performances of GA-SS and VNS-SS are near to GA and VNS, although GA-SS and VNS-SS have near zero WT, in both static FJSP and DFJSP.

This paper has only applied the idea of SS in FJSP and DFJSP, designed FJSP algorithms based on GA and VNS and demonstrated the effectiveness. In the future, more different EAs can be adopted in SS and SS can be developed to solve more scheduling problems other than FJSP.

Acknowledgment

The author would like to thank SJTU Innovation Center of Producer Service Development, Shanghai Research Center for industrial Informatics, Shanghai Key Lab of Advanced Manufacturing Environment, National Natural Science Foundation of China (Grant No. 71632008) and Major Special Basic Research Projects for Aero engines and Gas turbines (Grant No. 2017-I-0007-0008, Grant No.2017-I-0011-0012) for the funding support to this research.

References

[1] Gao K, Cao Z, Zhang L, et al. A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems[J]. IEEE/CAA Journal of Automatica Sinica, 2019, 6(4): 904-916.

[2] Mohan J, Lanka K, Rao A N, et al. A Review of Dynamic Job Shop Scheduling Techniques[J]. Procedia Manufacturing, 2019: 34-39.

[3] Pezzella F, Morganti G, Ciaschetti G, et al. A genetic algorithm for the Flexible Job-shop Scheduling Problem[J]. Computers & Operations Research, 2008, 35(10): 3202-3212.

[4] Amiri M, Zandieh M, Yazdani M, et al. A variable neighbourhood search algorithm for the flexible job-shop scheduling problem[J]. International Journal of Production Research, 2010, 48(19): 5671-5689.

[5] Bierwirth C, Kopfer H, Mattfeld D C, et al. Genetic algorithm based scheduling in a dynamic manufacturing environment[C]. ieee international conference on evolutionary computing, 1995.

[6] Bierwirth C, Mattfeld D C. Production scheduling and rescheduling with genetic algorithms[J]. Evolutionary Computing, 1999, 7(1): 1-17.

[7] Holthaus O. Scheduling in job shops with machine breakdowns: an experimental study[J]. Computers & Industrial Engineering, 1999, 36(1): 137-162.

[8] Rossi A, Dini G. Dynamic scheduling of FMS using a real-time genetic algorithm[J]. International Journal of Production Research, 2000, 38(1): 1-11.

[9] Ziaee M. A heuristic algorithm for solving flexible job shop scheduling problem[J]. The International Journal of Advanced Manufacturing Technology, 2014: 519-528.

[10] Rajabinasab A, Mansour S. Dynamic flexible job shop scheduling with alternative process plans: an agent-based approach[J]. The International Journal of Advanced Manufacturing Technology, 2011, 54(9): 1091-1107.

[11] Li X, Gao L. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem[J]. International Journal of Production Economics, 2016: 93-110.

[12] Gao J, Sun L, Gen M, et al. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems[J]. Computers & Operations Research, 2008, 35(9): 2892-2907.

[13] Gao K, Yang F, Zhou M, et al. Flexible Job-Shop Rescheduling for New Job Insertion by Using Discrete Jaya Algorithm[J]. IEEE Transactions on Systems, Man, and Cybernetics, 2019, 49(5): 1944-1955.

[14] Brandimarte P. Routing and scheduling in a flexible job shop by tabu search[J]. Annals of Operations Research, 1993, 41(1): 157-183.