A Dynamic Differential Evolution Algorithm for Dynamic Hybrid Flow Shop Scheduling Problem

Yue Zhao[a], Wanlei Wang[*, b], Jingping Yang[c] and Shikuan Zhou[d]

College of Mechanical and Electronic Engineering, Dalian Nationalities University, Dalian 116600, China.

[a]zhaoyue@dlnu.edu.cn, [b]wwl@dlnu.edu.cn, [c]yangjingping@gmail.com, [d]zskuan@163.com

Keywords: Differential evolution, hybrid flow shop, dynamic scheduling.

Abstract. In most real-world industries, scheduling is processed in a stochastic and dynamic environment, it is necessary to generate a schedule which is suitable for the current system states. Dynamic scheduling solves unimplemented jobs and updates an existing schedule based on the real-time information with minimizing the deviation between the new and original schedules. In this paper, we investigate the dynamic hybrid flow shop scheduling problem and propose a dynamic differential evolution algorithm. In the algorithm, the search space shifts as new jobs arrive, and the problem is solved on a moving horizon based on the real-time information, after the new schedule is established, we update the current schedule. Experiments are carried out to prove the effectiveness of the proposed algorithm.

Introduction

Hybrid flow shop scheduling (HFS) problem is one of the well-known scheduling problems. It is often found in various real-world industries such as manufacturing industry. There are many static scheduling techniques of HFS have been reported in the literature [1, 2]. In static scheduling, all the information is available initially and it does not change over time.

However, real scheduling problems are usually dynamic and are subject to various unexpected disturbances. These disturbances will upset the plan and cause original schedule becoming poor and sometimes unfeasible. Consequently, dynamic scheduling is needed to find a new schedule efficiently, quickly and keeping the production continuity. It is a challenging research area [3]. Few research works have addressed the problem. Ouelhadj and Petrovic [4] gave a good summary of the currently developing research on dynamic scheduling in manufacturing systems. Since scheduling in HFS of two or more stages tend to be NP-hard in general, scheduling dynamic HFS (DHFS) problem is also difficult. Kadipasaoglu *et al.* [5] gave a comparison of sequencing rules in static and dynamic hybrid flow system. Tang *et al.* [6] proposed a neural network model and algorithm for the HFS with dynamic job arrivals.

In recent years, differential evolution (DE), a stochastic population-based heuristic invented by Storn and Price [7], has a gradually development. Since it has simply model, good convergence, little parameter and easy implement, this algorithm has been successfully applied on many numerical optimization problems [8] and on many complex optimization problems [9-11].

In this paper, we propose a dynamic DE algorithm for the DHFS problem. In the proposed algorithm, real-time information is obtained continually, and the search space shifts as new jobs arrive, and then we get a new schedule based on the current schedule states, requirements, and real-time disturbances by the DE algorithm and update the current schedule.

Problem Formulation

The HFS problem we considered can be defined as follows. A set N of independent jobs, $N=\{1,\cdots,n\}$, need to be processed at a set of S stages in the same order, $S=\{1,\cdots,s\}$, starting at stage 1 until finishing in stage s. Each stage j, $j\in S$ composes of M_j unrelated parallel machines. A machine can process at most one job at a time and a job can be processed by at most one machine at a time. The preemption of such a processing is not allowed. Each job has a given processing time at each stage, a release date at the first stage, and a weight. The objective is to find a schedule that minimizes the total weighted completion time of the jobs.

To formulate the problem, the following notation is required.

p_{ijk}	the processing time of job *i* in stage *j* at machine *k*, $i \in N$, $j \in S$, $k \in M_j$
r_i	the release date of job *i*, that is, job *i* can not start processing before r_i, $i \in N$
w_i	the weight of job *i*, $i \in N$
S_{ij}	the start time of job *i* in stage *j*, $i \in N$, $j \in S$
C_{ij}	the completion time of job *i* in stage *j*, $i \in N$, $j \in S$

$$x_{ijk} = \begin{cases} 1 & \text{if job } i \text{ on stage } j \text{ is scheduled in machine } k, \\ 0 & \text{otherwise} \end{cases}$$

$$y_{iljk} = \begin{cases} 1 & \text{if job } i \text{ is scheduled immediately before job } l \text{ on machine } k \text{ at stage } j, \\ 0 & \text{otherwise} \end{cases}$$

With the above notation the HFS scheduling problem under consideration can be formulated as follows.

$$\min \sum_{i=1}^{N} w_i C_i \tag{1}$$

subject to

$$\sum_{k=1}^{M_j} x_{ijk} = 1, \quad i \in N, \ j \in S \tag{2}$$

$$\sum_{k=1}^{M_j} \sum_{i \in N} y_{iljk} \le 1, \quad l \in N, \ j \in S \tag{3}$$

$$S_{i1} \ge r_i, \quad i \in N \tag{4}$$

$$C_{ij} = S_{ij} + \sum_{k=1}^{M_j} p_{ijk} x_{ijk}, \quad i \in N, \ j \in S \tag{5}$$

$$S_{i,j+1} \ge C_{ij}, \quad i \in N, \ j \in S \tag{6}$$

$$\left(C_{ij} - S_{lj}\right) y_{iljk} \le 0, \quad i,l \in N, \ j \in S, \ k \in M_j \tag{7}$$

$$r_i \ge 0, \quad i \in N \tag{8}$$

$$x_{ijk} \in \{0,1\}, \quad i \in N, \ j \in S, \ k \in M_j \tag{9}$$

$$y_{iljk} \in \{0,1\}, \quad i,l \in N, \ j \in S, \ k \in M_j \tag{10}$$

Equation (1) in the above model describes the objective function. Constraint (2) guarantees that all jobs only can be processed at one machine in each stage. Constraint (3) ensures that at most one job immediately before another one on the same machine. Constraint (4) applies only to stage one, saying that a job cannot start its processing before its release date. Constraint (5) defines the relationship between the starting time and the completion time of a job. Constraint (6) ensures that for contiguous stages of the same job, only when the preceding operation is completed, the immediately next operation can be started. Constraint (7) ensures that for two contiguous jobs processed on the same machine, only when the preceding job is finished, the immediately next one can be started. Constraints (8), (9) and (10) define the domains of the variables.

Based on the above static scheduling model, we use a graph to present DHFS model, as in Fig. 1, which mainly consists of global scheduler, disturbance detector and dynamic controller. Normally, global scheduler obtains the static schedule by solving HFS. In a dynamic environment, disturbance detector gets the real-time disturbances, dynamic controller continually receives the real-time information and then controls the global scheduler dynamically and finally gets a new schedule.

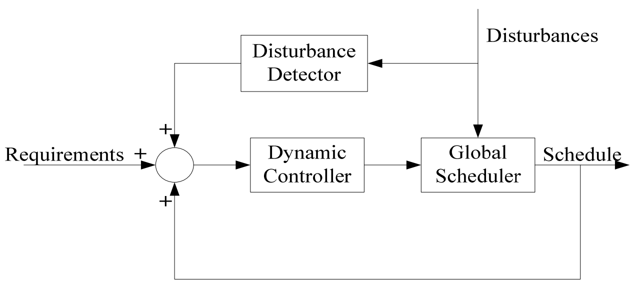

Fig. 1 The graph of DHFS processing model

Dynamic Differential Evolution Algorithm

The Overall Structure of the Algorithm. We consider our problem using a moving horizon presented in [12]. The dynamic scheduling problem is solved on the horizon $\left[t_0, t_0+L\Delta t\right]$, where L is a large enough number. At $t_0+\Delta t$, the scheduling horizon is shifted by Δt and the scheduling problem is solved again.

At t_0 we obtain all the information including requirements, disturbances, and schedule states, and then update the schedule states after a period of Δt and re-initializing the problem, new information is added to the optimization problem. In general, a new schedule solution is obtained. The flow chart of the dynamic DE algorithm is shown as Fig. 2.

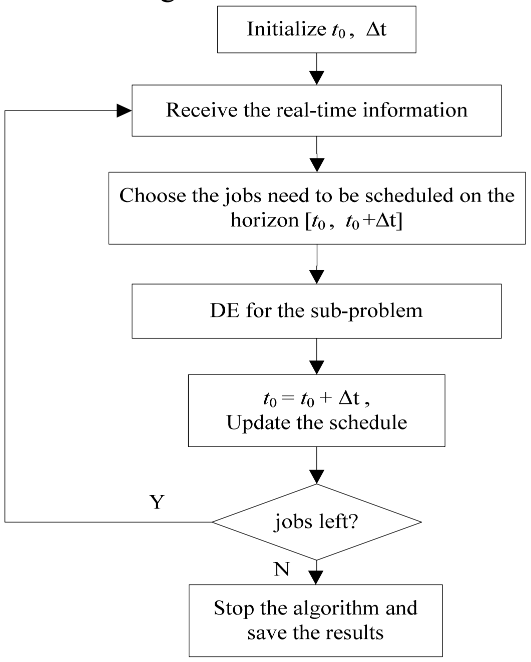
Initialize t_0, Δt
Receive the real-time information
Choose the jobs need to be scheduled on the horizon [t_0, t_0+Δt]
DE for the sub-problem
$t_0 = t_0 + \Delta t$, Update the schedule
jobs left?
Y
N
Stop the algorithm and save the results

Fig. 2 The flow chart of the dynamic DE algorithm

The Proposed DE Algorithm. DE is a population-based, derivative-free function optimizer, which utilizes *NP* *D*-dimensional parameter vectors $\mathbf{x}_{i,G} = \left(x_{1,i,G}, x_{2,i,G}, \cdots, x_{D,i,G}\right)$, $i = 1, 2, \ldots, NP$ as a population for each generation *G*. The population can also be written as $\mathbf{P}_G = (\mathbf{x}_{1,G},\ \mathbf{x}_{2,G},\ \cdots,\ \mathbf{x}_{NP,G})$ .In the proposed DE, it starts with the random initialization of a population of individuals in the search space, and then enters a loop of evolutionary operations: mutation, crossover and selection. The algorithm is terminated when a predefined maximum number of iterations is reached. The process of our DE algorithm is introduced as follows.

1) Individual representation

DE usually encodes decision variables as floating-point numbers. For the HFS problem, we use a real-coded matrix which reflects the assignment of jobs to machines at every stage as an individual of the population. The individual matrix $\mathbf{A}_{m\times s}$ is given by:

$$\mathbf{A}_{m\times s}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{ms} \end{bmatrix}$$

(11)

where a_{ij} $,i\in N$, $j\in S$, which is a randomly floating-point number and $a_{ij}\in(1,M_j+1)$, denotes that job *i* in stage *j* is scheduled on machine $\lfloor a_{ij} \rfloor$.

2) The decoding scheme

Present researches on solving the HFS problem usually firstly fix the job allocation and job permutation and secondly establish the job time table. In this paper, DE algorithm is used to fix the job allocation, and then apply the FIFO rule to establish the job permutation and the job time table under sequencing the jobs according to the ascending order of release dates in stage 1. The FIFO rule denotes that the job will be firstly scheduled if it firstly arrives.

3) Evolutionary operations

In the algorithm, we adopt standard crossover and selection operators, but propose a new mutation operator. At each generation *G*, this operation creates mutation vector $\mathbf{v}_{i,G+1}$ based on the current parent vector $\mathbf{x}_{i,G}$, $i=1,2,\ldots,NP$. Inspired by Tang *et al.* [11], we introduce a memory scheme in the mutation strategy to improve the performance. Let $\mathbf{P}^{\mathrm{M}}$ denote the memory population which immediately precedes the current population $\mathbf{P}$. The following is the mutation strategy:

$$\mathbf{v}_{i,G+1}=\mathbf{x}_{r_1,G}+F(\mathbf{x}_{r_2,G}-\mathbf{x}_{r_3,G})+F(\mathbf{x}^{M}_{r_4,G}-\mathbf{x}^{M}_{r_5,G}),\ i\neq r_1\neq r_2\neq r_3,\ r_4\neq r_5 \qquad (12)$$

where indexes r_1, r_2 and r_3 are randomly integers and $r_1,r_2,r_3\in\{1,2,\cdots,NP\}$. $\mathbf{x}^{M}_{r_4,G}$ and $\mathbf{x}^{M}_{r_5,G}$ are two different individuals randomly chosen from $\mathbf{P}^{\mathrm{M}}$. The mutation factor $F\in(0,1]$.

Computational Experiment

The disturbances we tested are machine failures, job processing time changes, and dynamic job arrivals. And the other disturbances can be also solved. The proposed dynamic DE algorithm (DDE) is compared with conventional DE algorithm.

In our experiments, the parameters are set as follows: *NP*=50, *F*=0.5, *CR*=0.8, *G*=50, $t_0=0$, $\Delta t=20$. Every group data was run 30 times and we saved the average value. Test problems are generated as follows:

(1) Number of jobs $n\in\{10,30,70,100\}$.

(2) Number of stages $s\in\{2,3\}$.

(3) Number of machines at each stage is 3.

(4) The weights are integers uniformly drawn from [1, 10] , the processing times from [1, 50], and the arrival times from $[t_0, t_0+50]$.

(5) The machine failures are randomly generated.

(6) The job processing time changes according to 10% probability, which means 10% probabilities jobs will change the processing time. The change time is integer uniformly drawn from [10, 20].

According to the above, we generated eight different problems for testing. Fig. 3 shows the graph of the objective function value as a function of the generation number for the 10×2×3-problem data set. We can see that the algorithm is converged.

To evaluate the performance of our proposed DDE algorithm, we apply the utility and stability measures presented in the literatures [13, 14]. The experimental results on the performance of the utility and stability measures of the proposed DDE algorithm and the conventional DE algorithm are presented in Table 1 and Fig. 4.

From the results from Table 1 and Fig. 4, we can see that the DDE algorithm has a consistent behavior and a better performance in both the utility and stability measures for all instances compared to DE algorithm. The DDE algorithm can provide satisfactory dynamic schedules with only 3.41% deviation in objective and 2.83% deviation in the scheduled time of jobs. Moreover, our algorithm can even improve some original schedules.

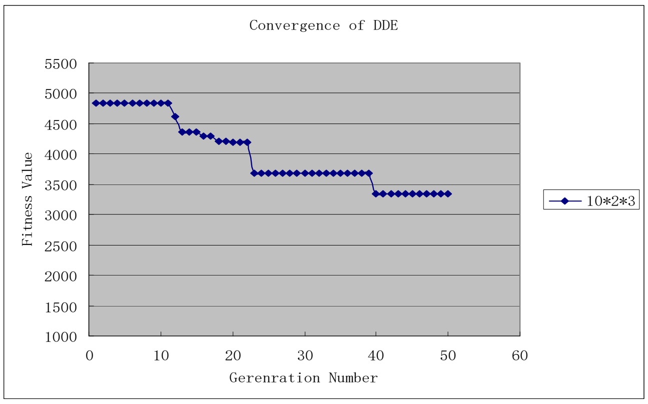

Fig. 3 The graph of DDE performance

Table 1 Comparison Result Between DE and DDE

P	Structure[a]	DE		DDE	
		Utility	*Stability*	*Utility*	*Stability*
1	10×2×3	0.0615	0.0512	0.0606	0.0265
2	10×3×3	-0.2888	0.0914	-0.0618	0.0426
3	30×2×3	-0.2716	0.0901	-0.0162	0.0078
4	30×3×3	-0.2669	0.1400	-0.0882	0.0234
5	70×2×3	-0.1357	0.0951	-0.0293	0.0173
6	70×3×3	-0.2614	0.1638	-0.0597	0.0447
7	100×2×3	-0.1199	0.1029	-0.0189	0.0179
8	100×3×3	-0.2905	0.2193	-0.0590	0.0461
Average		-0.1967	0.1192	-0.0341	0.0283

[a]Structure means Jobs×Stages×Machines, the value of machines denotes that there is the same machine number in each stage.

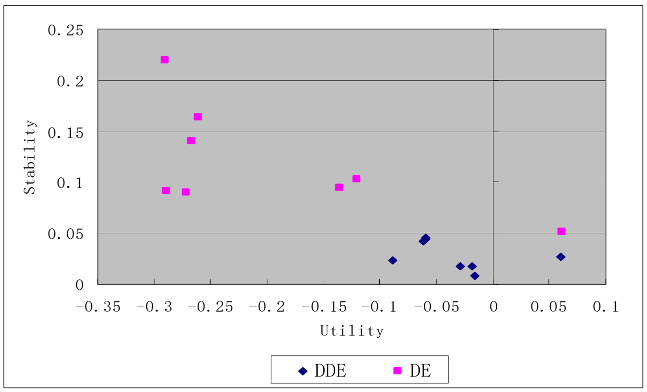

Fig. 4 Performance of utility and stability measures

Conclusion

This paper investigates a dynamic scheduling for hybrid flow shop scheduling problem with unrelated parallel machines and presents a dynamic differential evolution algorithm for it. Because of the complexity of the problem, we give a mathematical model for the static scheduling problem, and then give a graph model for the dynamic scheduling problem. Considering the real-time requirements, we apply a dynamic optimization strategy in which the real-time information including disturbances, the requirements, and the schedule states is continually obtained in a moving horizon, and then DE algorithm is used to solve the sub-problem on a scheduling horizon. It is observed from the experiment that dynamic different evolution algorithm can efficiently deal with the disturbances and have a consistent behavior and a better performance in both the utility and stability measures for all instances compared to conventional DE algorithm.

Acknowledgments

*Wanlei Wang is the corresponding author. This research is supported by Research Foundation of Liaoning Educational Committee (L2013511) and Self-Topic Foundation of Dalian Nationalities University (DC201502010301).

References

[1] I. Ribas, R. Leisten, and J.M. Framinan, Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective, Computers & Operations Research 37 (2010) 1439-1454.

[2] R. Ruiz and J.A. Vazquez-Rodriguez, The hybrid flow shop scheduling problem, European Journal of Operational Research 205 (2010) 1-18.

[3] L.X. Tang, J.Y. Liu, A.Y. Rong, and Z.H. Yang, A review of planning and scheduling systems and methods for integrated steel production, European Journal of Operation Research 133 (2001) 1-20.

[4] D. Ouelhadj and S. Petrovic, A survey of dynamic scheduling in manufacturing systems, Journal of Scheduling 12 (2009) 417-431.

[5] S.N. Kadipasaoglu, W. Xiang, and B.M. Khumawala, A comparison of sequencing rules in static and dynamic hybrid flow systems, International Journal of Production Research 35 (1997) 1359-1384.

[6] L.X. Tang, W.X. Liu, and J.Y. Liu, A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment, Journal of Intelligent Manufacturing 16 (2005) 361-370.

[7] R. Storn and K. Price, Differential evolution – a simple and efficient heuristic strategy for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341-359.

[8] P. Kaelo and M.M. Ali, A numerical study of some modified differential evolution algorithms, European Journal of Operational Research 169 (2006) 1176-2284.

[9] G. Onwubolu and D. Davendra, Scheduling flow shops using differential evolution algorithm, European Journal of Operation Research 171 (2006) 674-692.

[10] A.C. Nearchou, A differential evolution approach for the common due date early/tardy job scheduling problem, Computer & Operations Research 35 (2008) 1329-1343.

[11] L.X. Tang, Y. Zhao and J.Y. Liu, An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production, IEEE Transactions on Evolutionary Computation 18 (2014) 209-225.

[12] J. Busch, J. Oldenburg, M. Santos, A. Cruse, and W. Marquardt, Dynamic predictive scheduling of operational strategies for continuous processes using mixed-logic dynamic optimization, Computers and Chemical Engineering 31 (2007) 574-587.

[13] D. Quelhadj, P.I. Cowling, and S. Petrovic, Utility and Stability Measures for Agent-Based Dynamic Scheduling of Steel Continuous Casting, in 2003 IEEE International Conference on Robotics and Automation (2003) 175-180.

[14] P.I. Cowling, M. Johansson, Using real time information for effective dynamic scheduling, European Journal of Operation Research 139 (2002) 230-244.