Greedy Algorithm Solution of Flexible Flow Shop Scheduling Problem

Xiaofeng Li *Hai Zhao*

College of Information Science and Engineering, Northeastern University, Shenyang, Liaoning Province, 110004

Business Support Center of China Mobile Group Jilin Co., Ltd., Changchun, Jilin Province, 130021

Abstract: Flexible flow shop scheduling problem (also called blended flow shop scheduling problem) is a sort of complex Job Shop Scheduling problem. For the Flexible Flow Shop scheduling problem, only in very special cases, there are polynomial optimal algorithms. In most of the other cases, the problems are NP-Hard. It is a simplification of the original problem to solve Flexible flow shop scheduling with Greedy algorithm, and it is also a combination of efficiency and algorithm. In this paper, a greedy algorithm solving flexible flow shop scheduling problem is given, and the capability of the algorithm is evaluated.

Key words: Flexible Flow Shop, Job Shop Scheduling, Greedy Algorithm

1. Introduction

Scheduling problem [1] was first proposed for machine manufacturing industry and later it has been widely applied in fields such as computer system, vehicle scheduling, production management, etc. The theory and algorithm of scheduling are used from daily life schedules such as production schedule, personnel schedule and curriculum schedule and to the huge and complex flight plan of spacecraft.

As extensions of classical schedule problems, schedule problems of processors also have wide background. There are mainly following two factors which promoted the study on the solutions of Flexible Flow Shop scheduling problem.

1) Reasonable and efficient schedule can bring huge economic benefits in manufacturing and there is no need to consume excessive physical resources.
2) In the normal cases, schedule problems are HP-Hard problems which are difficult to solve, and therefore, it is of import academic value to study scheduling problems.

Multiple discussions can be adopted for the situation that the objective function is minimized scheduling table length. At present, the corresponding solutions such as mathematical models including mixed integer model [2], Flow2Shop model [3] and immune genetic algorithm, etc. are proposed.

2. Problem Description

2.1 Definition of Scheduling Problem

Scheduling problem is a class of important combinatorial optimization problem where the given Task or Job is optimally finished by use of Processors, Machines or Resources [1-3]. During the execution of these tasks or jobs, some limited conditions should be met, that is, the objective function, which is a description of the length of processing time and processor utilization, should reach the minimum value.

Task and Job: they are the constraint conditions in the scheduling problem. They mainly refer to the nature of the Tasks and Jobs and the requirements and constraints for them in the processes. The processing time vector of the job is: $Tj=(t_{1j}, t_{2j}, \ldots\ldots, t_{nj})$, where t_{ij} is the processing time when Job j is processed in the processing center i.

Arrival time or Ready time: r_j refers to the time when Job j is ready for being processed. If the Ready time of all the Jobs are the same, we take $r_j=0$, j=1,2,……,n.

Due date: d_j refers to the restricted completion time for Job j. The time limit is called Deadline.

Priority Factor: w_j is weight, which indicts the importance degrees of Job j.

Processor: Scheduling problems in which there is only one processor are called Single Processor scheduling problem, otherwise the problems are called multiprocessor scheduling problem.

For multiprocessor scheduling problems, if all the processors perform identical function, they are called as **Uniform processors** or **Parallel processors**.

If every job is needed to be processed by all the processing centers, that is $n_j=m$, j=1,2,…,n, and each job is processed with the same processes in each processing center, this kind of problem is called as Permutation Flowshop Scheduling or **Flow shop**.

Manuscript received November 5, 2009
Manuscript revised November 20, 2009

Objective function: we use C=(C_1, C_2, ……, C_n) to denote Completion time and the objective functions to be minimized are the functions of the Completion time C_j for the job[1]. In the scheduling problems, there are following types of objective functions.

Schedule length: Its definition is $C_{max}=\max\{C_j\}$ which equals to the completion time for the job which is last completed. Small schedule length means high utilization of processors.

Mean weighted flow time is:

$$F = \sum w_j F_j / \sum w_j$$

where $F_j = C_j - r_j$ is the flow time of Job j, which equals to the sum of waiting time and processing time the job spends in the system.

Deterministic scheduling problem: Among the scheduling problems, if all the data is know before decisions are made, such kind of scheduling problems are called as **Determinist scheduling problem**[3][4].

2.2 Job Shop Scheduling Problem

Job shop scheduling problem mainly includes Permutation Flowshop Scheduling Problem, Open-Shop Scheduling Problem and asynchronous job scheduling problem. As to the job shop scheduling problem, there are no polynomial algorithms except a few ones. Many people provide heuristic solutions to job shop scheduling problems, for example: Artificial Immune Algorithm for Flow-Shop Scheduling [2], Solution of jobshop scheduling problems based on evolutionary algorithms [3] and a genetic descent algorithm for hybrid flow shop scheduling [5], and etc.

Permutation Flowshop problem is also called Job Shop Scheduling problem, which can be expressed as: Fm||g, where g is a non-decreasing function of completion time. It is a common and important kind of job shop scheduling problem. The common objection functions are minimized scheduling length. Most of flowshop problems are NP-Hard.

In the flowshop scheduling problems, all the jobs are processed by processors P_1, P_2, ……, P_m in turn. However, the processes for jobs done by the same processor may be different. If the processes of any job done by all processors are identical, it is called Permutation schedule. The schedule number of Permutation Flowshop is $(n!)^m$ for n jobs and m processors. If we only consider Permutation schedule, the schedule number is n!. However, the optimal scheduling for Permutation Flowshop Scheduling are not always Permultaion schedule when $m \geq 4$.

2.3 Flexible flow shop scheduling problem

Flexible flow shop scheduling problem (also called blended flow shop scheduling problem) [5] is a sort of complex Job Shop Scheduling problem. In the Flexible flow shop scheduling problem, let Z_1, Z_2, ……, Z_s be the S processing centers, among which there are m_L synchronizers at No. L processing center Z_L and every two processing centers have unlimited storage capacity between them. The n jobs are : J_1, J_2, ……, J_n and for job J_j, there are s processes which are T_{1j}, T_{2j}, ……, T_{sj} and the processing time for process T_{Lj} is t_{Lj} where $t_{Lj} \geq 0$, L=1,2,……,s, and j=1,2,……,n. The process T_{Lj} can be processed by every processor of processing center Z_L.

This problem is often denoted by FFs | m_1,m_2,…,m_s |g where g denotes the non-decreasing function of completion time for the job.(Figure 1)

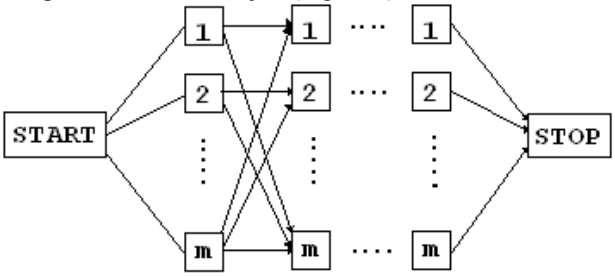

Figure 1 Flexible flow shop scheduling problem

Apparently, if there is only one processor in every processing center, the problem is transformed into Permutation Flowshop problem. If there is only one processing center, the problem is transformed into parallel machine scheduling problem. Hence, flexible flow shop scheduling problem is an extension of parallel machine scheduling problem and Permutation Flowshop problem. For the Flexible Flow Shop scheduling problem, only in very special cases, there are polynomial optimal algorithms. In most of the other cases, the problems are NP-Hard.

3. Greedy Algorithm Solution of Flexible Flow Shop Scheduling Problem

3.1 Model of Simple Flexible Flow Shop Scheduling Problem

It is a simplification of the original problem to solve Flexible flow shop scheduling with Greedy algorithm, and it is also a combination of efficiency and algorithm. If solving NP-Hard problem is for mathematics study, the optimal solution is purpose, no matter how much the time cost is; however, if solving NP-Hard problem is for

computer study, the optimal solution may be not the purpose, while the relative optimal solution and less time complexity is the purpose.

Therefore, we can adopt some common algorithms to simulate solving Flexible flow shop scheduling for study purpose so that to make the time complexity be polynomial time or linear time. It is a good choice to adopt Greedy Method to solve Flexible flow shop scheduling

After further abstraction of Flexible flow shop scheduling, a simple model for Job shop scheduling is obtained.

We define that, the Simple Flexible Flow Shop (hereafter referred to as SFFS for short) has m processing centers which are Z_1, Z_2, ……, Z_m where processing center Z_j have z_j parallel processors; there are n jobs, and each job must be processed by any processor of processing centers Z_1, Z_2, ……, Z_m in order and the processing time of job in the processing center is vector J_j, where $J_j=(j_1, j_2, ……, j_m)$.

The constraints are as follows:

a) On processing center can only process one job at the same time(that is, time shouldn't be overlapped)
b) One processing center consists of one or many parallel processors, all of which can work at the same time, processing different jobs;
c) All the processes of each job must be done in order, and a process shouldn't be started until its pervious process is finished and one process can be done by any parallel processor of the corresponding processing center.

Based on the three constraints above, it can be concluded that, if we adopt Greedy Method to solve Flexible flow shop scheduling, the following requirements must be met:

1. Every processor must record a last completion time (that is, the completion time when the processor has finished its last job which has been allocated to it) so that the jobs allocated later can be processed immediately when the pervious job is finished.

2. Every job must also record its last completion time in its pervious process to ensure the beginning time for process must equals to or later than completion time of the previous process.

3.2 Greedy Algorithm and Analysis

In conclusion, we provide the following Greedy solving strategies:

a) Let the current processing center be the first processing center, the current processor be the first processor of the processing center and the current process of the job be the first process.
b) Choose the job which is finished earliest currently and place it into the current processor and then let the current processor be the next processor of the current processing center; Repeat the processes until the placement of all the jobs are finished, and then start the next process.
c) Let the process be the next process. Return to b) and execute again until all the processes are finished. The algorithm is finished.

The pseudocode of this process is as follows:

```
    //Completion time for the last process of the
      job
    int je[JOB_MAX]={0};
    // Final completion time of the processor
    int pp[20];
    for (every process i of the job) {
       Assign 0 to all the pp;
       Schedule the Jobs according to their
       completion times
          int k=0; //  the No.k processor.
          for (all the jobs j after scheduling) {
                    int pos=pp[k] and the last ones
       among the completion time of the job;
                    //Place Job to position pos of
       processor k in the processing center i
                    PositionJob(k, job, pos);
                    pp[k]=pos+ processing time for
       job j;
                    k++;
          } // end of for j
} // end of for i
```

Suppose there are n jobs, each of which has m processes, that is, there are m processing centers, each of which has z_1, z_2, ……, z_m processors and the processing time for each job at every processing center is t_1, t_2, ……, t_n and then the time complexity of Greedy Algorithm is

$$m * (c + s + n) \quad \text{…… (1)}$$

where c is a constant, which is mainly determined by Z, the number of parallel processors of all the processing centers; s is the time complexity of the scheduling algorithm. If the scheduling algorithm adopt quick sorting, the expected time complexity is $O(n*\log_2 n)$ and the equation (1) transformed into :

$$m * (c + s + n) = O(m*n) \quad \text{……(2)}$$

Therefore, the time complexity for Greedy Algorithm is O(m*n).

In the common productions, the number of processing centers is not more than 10, each of which consists of 5 processors or less and the number of jobs is commonly about 10-20. In this way, m=10, n=20, Z = 5+ 5+ ……+5=50, the scheduling time is about 86, and therefore, the time of Greedy Algorithm is about 10*(1+86+20) = 1070.

Apparently, though there are big difference between the approximate solutions (relative Optimal Solution) acquired by Greedy Algorithm and the optimal solutions, however, it is enough for studying Flexible flow shop scheduling problem due to its low time complexity. What's more, there is network expansion for further study of Flexible flow shop scheduling problem and it can be applied for the solution with remote heuristic algorithm later.

4. Simulation Experiment

The type of simulation program is Win32 Application with MFC [5] support, the development environment is Windows XP + Visual Studio .NET 2003[5]. The whole project is designed according to MVC [6][7] (Module View Controller).

Test Environment:

Intel P4 1.7G Hz CPU

512M DDR MEMORY

Microsoft Windows XP

Table 1 Simulation data

Input Size		Running Time of Algorithm(sec)
Process (m)	Num of Jobs (n)	
10	8	<0.1
10	16	0.1
10	20	0.2

The test sizes above are all similar to the data of practical production.

5. Conclusions

Because of the complexity of Flexible flow shop scheduling problem (FFS), it is hard for small-size optimal solution to this kind of problem and it is almost infeasible for large-size optimal algorithm. This paper introduces the Flexible flow shop scheduling problem, which is simplified into SFFS (Simple FFS) problem for simulation implementation. Then we adopt Greedy Algorithm to simulate approximate solution. Though there is difference between the approximate salutation and theoretical solution, it can be adopted for studying this kind of problem and small-scale production due to its extremely low time complexity. To the FFS problem, there are other algorithms, such as Evolutionary algorithm for solving multi - objective hybrid flow- shop scheduling problem [8], Particle Swarm Optimization Algorithm in Flexible Job Shop Scheduling Problems [9], Application of An Improved Genetic Algorithm for Shop Floor Scheduling [10], and etc, which should be studied in our future works.

References

[1] Tang HY , Zhao CL , Scheduling Introduction[M] , Beijing, Science Publishing Press, 2002

[2] Zhang J, Li P, Solution of job shop scheduling problems based on evolutionary algorithms[J], Journal of Zhejiang University, 2004.38(12), pp: 1545-1549

[3] Wang ZQ, Feng BQ, Artificial Immune Algorithm for Flow-Shop Scheduling [J], Journal of Xi'an Jiaotong University, 2004.38(10), pp: 1031-1034

[4] Huang XY, Zhang ZH, He CJ, etc. Modern Intelligent Algorithm Theory and Application [M], Science Publishing Press, 2005

[5] Tang LX, Wu YP, A genetic descent algorithm for hybrid flow shop scheduling [J], Journal of Automation, 2002.28(7), pp: 637-641

[6] Hou J, Dessecting MFC [M], Huazhong University of Science and Technology, 2001

[7] Charles Petzold, Programming Windows [M], Microsoft Press, 2005

[8] Wei Z, Xu XF, Deng SC, Evolutionary algorithm for solving multi - objective hybrid flow- shop scheduling problem [J], Computer Integrated Manufacture System, 2006,12(8), pp: 1227-1234

[9] Jia ZH, Application and Research on Particle Swarm Optimization Algorithm in Flexible Job Shop Scheduling Problems [D], Phd thesis of university of science and technology of China, 2008

[10] Wang T, Fu YL, Application of An Improved Genetic Algorithm for Shop Floor Scheduling [J], Computer Integrated Manufacture System, 2002,8(5), pp: 392-395,420

Li Xiaofeng, born in 1967, Ph.D., student of Northeastern University. The main research field is Financial Computer Security. Implemented and completed the sub-project of the *Research and Demonstration on Framework for Security Safeguarding System of National Important Financial Information System*, which is the project of National High-tech R&D Program (863 Program).

Zhao Hai, Male, Professor of Northeastern University doctoral tutor Director of Research Lab of Embedded Technology System. The main research Field is Computer Network Communication. Led the projects of National High-tech R&D Program (863 Program) and National Torch Program.