The distributed permutation flowshop scheduling problem: A genetic algorithm approach

Yan Li[1, a *], Zhigang Chen[2,b]

[1]Department of Industrial Engineering, Shanghai Second Polytechnic University, China

[2]F2Department of Logistics, Shanghai Second Polytechnic University, China

[a]liyan@sspu.edu.cn, [b]zgche@sspu.edu.cn

Keywords: Distributed permutation flowshop scheduling;Genetic algorithms;Design of experiments.

Abstract. We consider solving a distributed permutation flowshop scheduling problem (DPFSP) with the objective of minimizing makespan. The problem has two dimensions: assigning jobs to factories and scheduling the jobs assigned to each factory. We use GA to solve this problem. In the proposed algorithm we employ some standard techniques like one point crossover and swap mutation. Computational experiments show that the proposed GA produces improved results than original ones but not powerful enough to produce better ones than the known best solutions. Based on this study we will redesign the standard GA implementation by using structural information from the problem. Combining GA with constraint programming and other heuristics to design hybrid search algorithms will also be a valuable direction for further study.

Introduction

The distributed permutation flowshop problem (DPFSP) was recently proposed by Naderi and Ruiz [1]as a generalization of the regular flowshop setting where more than one factory is available to process jobs. The problem has two dimensions: assigning jobs to factories and scheduling the jobs assigned to each factory.

The first heuristics to solve the DPFSP was proposed by Naderi and Ruiz [1]. They suggested four constructive heuristics, denoted NEH1, NEH2, VND(a) and VND(b), following the ideas taken from the PFSP and employing Taillard's acceleration. More specifically, NEH1 and NEH2 are adaptations to the problem of the original NEH by means of using two assignment rules to choose the factory where a job has to be introduced were tried. Gao and Chen [2] were the first authors who proposed an iterated optimisation algorithm for the problem. Their proposal was later outperformed by the tabu search algorithm (TS) by Gao, Chen, and Deng [3]. Wang et al. [4] implemented an Estimation of Distribution Algorithm (EDA). Lin, Ying, and Huang [5] proposed a variation of the iterated greedy approach.

In this paper we present a genetic algorithm (GA) method for this problem to optimize makespan. We employ some standard techniques like one point crossover and mutation. Computational experiments show that the proposed GA algorithm produces improve results than original answers.

Problem description

The problem under consideration can be stated as follows: n jobs have to be scheduled in one of the F flowshop factories consisting of m machines. Each factory is identical with the same set of m machines and is able to process all jobs. Once a job is assigned to a factory, it has to be processed there without being transferred to another factory. On each machine i, each job j has a processing time denoted as p_{ij}regardless the factory f where the job is processed. The problem determines the sequenceπ^f, formed by $n_f$jobs, to be scheduled in each factory f. Therefore, a solution π is formed by the sequence in each factory ($\pi = [\pi^1, \pi^2, \dots, \pi^f, \dots, \pi^F]$). Let $C_{i,j}^f$ be the completion time of job j in machine i when assigned to factory f, and $C_{max}^f = C_{max}(\pi^f)$ the makespan of factory f. Then $C_{max} = C_{max}(\pi)$ denotes the global makespan, the completion time of the last job to be processed in any factory. Additionally, $\pi^f[i]$ is employed to denote the element of factory f in position i. By using

f_{max} to denote the factory with maximum makespan, the global makespan can be also written as $C_{max}^{f_{max}}$.

Genetic algorithm approach

We follow the standard GA method. A good reference for understanding how GAs work is Goldberg [6].The first step in applying GA to a particular problem is to convert the feasible solutions of that problem into a string type structure called chromosome. In order to find the optimal solution of a problem, a standard GA starts from a set of assumed or randomly generated solutions called initial population and evolves different but better sets of solution over a sequence of generations (iterations). In each generation the objective function (fitness measuring criterion) determines the suitability of each chromosome and, based on these values, some of them are selected for reproduction. For the distributed flowshop scheduling problem we take the fitness value of each chromosome to be the reciprocal of the makespan. The number of copies reproduced by an individual parent is expected to be directly proportional to its fitness value, thereby embodying the natural selection procedure, to some extent. The procedure thus selects better (highly fitted) chromosomes and the worse one are eliminated. Genetic operators such as crossover and mutation are applied to these (reproduced) chromosomes and new chromosomes (offspring) are generated. These new chromosomes constitute the next generation. These iterations continue till some termination criterion is satisfied.

Encoding scheme

In the PFSP literature, the most common solution representation is a permutation of the n jobs. Since in the DPFSP this permutation is divided among the F factories, the most straightforward representation is to have F lists, one per factory. Each list contains a partial permutation with the order in which the jobs have to be processed at each factory. This is the solution representation that [1]and subsequent authors have employed.

We present a different representation by constructing a priority value of sequence individual. An individual is represented by a vector λ of priority values. Each priority value in the vector λ is linked to a job and based on these values, the sequence in which the jobs and activities will be scheduled is assigned. For an example of 5 jobs, the individual can be depicted as

position: job IDs				
1	2	3	4	5
2	1	3	5	4
λ:priority values				

Fig.1 Priority value based encoding

Initial population

The initial population consists of N randomly generated job sequences. N is referred to as the population size.

Reproduction

Individual chromosomes from the present population are copied according to their fitness values. This is done by randomly selecting (with replacement) chromosomes with probability proportional to their fitness value. Roughly speaking, one would expect $N \times f_i$ number of copies of string i in the gene pool used to create the next generation. This forms the mating pool to which the crossover and mutation operators are applied to form the next generation.

Crossover

Two parents are selected from the mating pool at random. With probability $1 - p_c$, the parents are copied as they are. With probability p_c the following operation is performed: a single point crossover (1X operator) between two parents is carried out by choosing a number *k* at random between 1 and *l* − 1, where *l* is the length of the string (*l* >1). Two new strings are created. The first child is created by copying all characters of the chromosome of the first parent to location *k*. The remaining places are filled by scanning parent 2 from left to right and entering the priority values not already present. Child 2 is created in a similar way by reversing the roles of the parents.

Mutation

Mutation is done by selecting two different locations on the chromosome at random and interchanging the priority values at these locations. For each child obtained from crossover, the mutation operator is applied independently with a small probability p_m.

Factory assignment

The allocation dimension of the DPFSP can be added to the GA by including a factory assignment rule that is applied after sequencing. In this paper, we adopt NEH2[1] to allocate jobs to factories: Assign job *j* to the factory which completes it at the earliest time, i.e., the factory with the lowest C_{max}, after including job *j*.

Fitness evaluation function

In order to mimic the natural process of the survival of the fittest, the fitness evaluation function assigns to each member of the population a value reflecting their relative superiority (or inferiority). Let $r_i, i = 1, 2, \ldots, N$ denote the reciprocal of the makespan of the strings in the population. The fitness value assigned to string i would then be proportional to $f_i = r_i / \sum_j r_j$.

Termination criterion

In our implementation of GA, we terminate the algorithm after 20 generations.

Computational experiments

The proposed algorithm is implemented in C++ on a PC of Intel Core i3 with 3.30 GHz and 4.00 GB RAM. We use a set of standard test problems available in the web page for the research group Applied Optimization Systems (http://soa.iti.es/problem-instances). All instances, along with the best known solutions are available at http://soa.iti.es. These instances extend the regular flowshop instances with multiple factories and have been used in [1].

Number of factories	Number of jobs	Number of stages	Number of instances	Improvement rate (%)	Gap (%)
2	20	5	120	2.08	22.61
3	20	5	120	2.97	25.66
4	20	5	120	3.65	27.84
5	20	5	120	3.78	30.17
6	20	5	120	3.79	32.29
7	20	5	120	3.49	33.84
Total				3.29	28.73

Table 1: Solutions for DPFSP benchmark instances

Table 1 shows the results of computational experiments. We employ improvement rate and gap to express the effectiveness and efficiency of the proposed GA:

$$Improvement\ rate = \left(1 - \frac{final\ makespan}{original\ makespan}\right) \times 100\% \quad (1)$$

where original makespan is the first of five of the best values of makespan of each instance and final makespan, the best one of our answers, is the last of the best values. The total average improvement rate of 3.29% shows that the proposed GA can make improvements in makespan within 20 seconds and thus is effective.

$$Gap = \left(1 - \frac{optimal\ makespan}{final\ makespan}\right) \times 100\% \quad (2)$$

where optimal makespan is the best known solutions from http://soa.iti.es/problem-instances, and final makespan is the best one of our answers. The total average gap of 28.73% indicates that the proposed GA is not powerful enough to improve the currently best known solutions. However, we hope to design a hybrid algorithm based on the proposed GA to improve efficiency in the near future.

Conclusions

We consider solving a distributed flowshop problem with the objective of minimizing makespan. We use GA to solve this problem since it is difficult to obtain exact optimal solutions in a reasonable amount of time. GA has seldom been explored for distributed permutation flowshop settings. In the proposed algorithm we employ some standard techniques like one point crossover and swap mutation. Computational experiments show that the proposed GA produces improved results than original ones but not powerful enough to produce better ones than the known best solutions. For further research, we will redesign the standard GA implementation by using structural information from the problem. Using information about the problem structure seems essential for better use of search algorithms like GAs. Combining GA with constraint programming and other heuristics to design hybrid search algorithms will also be a valuable direction.

Acknowledgements

This work was financially supported by Innovation Program of Shanghai Municipal Education Commission（No. 13YS121）and Academic Development Program of Shanghai Second Polytechnic University (Management Science and Engineering, No. XXKPY1313).

References

[1] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Computers & Operations Research, 37 (2010) 754-768.
[2] J. Gao, R. Chen, A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem, International Journal of Computational Intelligence Systems, 4 (2011) 497-508.
[3] J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, International Journal of Production Research, 51 (2013) 641-651.
[4] S.Y. Wang, L. Wang, M. Liu, Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, International Journal of Production Economics, 145 (2013) 387-396.
[5] S.W. Lin, K.C. Ying, C.Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, International Journal of Production Research, 51 (2013) 5029-5038.
[6] D.E. Goldberg, GA in search, optimisation, and machine learning, Seventh Impression, Pearson, (1989).