A Hybrid Genetic Algorithm for the Distributed Permutation Flowshop Scheduling Problem

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

[1]Department of Industrial Engineering, Shanghai Second Polytechnic University, Shanghai, 201209, China

[2]Department of Logistics, Shanghai Second Polytechnic University, Shanghai, 201209, China

[a]email: liyan@sspu.edu.cn, [b]email: zgchen@sspu.edu.cn

Keywords: Distributed permutation flowshop scheduling; Genetic algorithms; Local search; Hybrid genetic algorithm.

Abstract. For solving the distributed permutation flowshop scheduling problem (DPFSP) with the objective of minimizing makespan we design a hybrid genetic algorithm. This algorithm combines a simple genetic algorithm, a local search algorithm and a plant allocation rule with the aim of minimizing the makespan. The experimental results show that the proposed algorithm outperforms the genetic algorithm in terms of efficiency and effectiveness for parts of the data.

Introduction

The distributed permutation flowshop problem was recently proposed by Naderi and Ruiz [1]as a generalization of the regular flowshop setting where more than one plant 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. Using NEH2 and VND(a) as initial solutions, Gao and Chen [2] were the first authors who proposed an iterated optimization 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 for the problem a variation of the iterated greedy, denoted MIG.

This paper presents a hybrid genetic algorithm (GA) method for this problem to optimize makespan. We employ some standard techniques like one point crossover and mutation combined with a local search and a plant assignment rule. Computational experiments show that the proposed 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 $p_{i,j}$regardless the factory f where the job is processed. The problem determines the sequence π^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 ($\pi = [\pi^1, \pi^2, \ldots, \pi^f, \ldots, \pi^F]$). 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. i.e. 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 $C_{max}^{f_{max}}$.

Propose hybrid genetic algorithm approach

The proposed algorithm can be summarized as Fig.1 follows.

***Step* 1.** Create initial population randomly.
***Step* 2.** Stopping criteria = false? If no go to step 8; otherwise continue.
***Step* 3.** Select survivor genes
***Step* 4.** Crossover
***Step* 5.** Mutation
***Step* 6.** Local search
Step 7. Plant assignment
***Step* 7.** Evaluate fitness
***Step* 8.** Stop HGA

Fig.1 Overall Hybrid GA framework

position: job IDs

1	2	3	4	5
2	1	3	5	4

λ: priority values

Fig.2 Priority value based encoding

Encoding scheme. 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.

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, called the crossover probability the following operation is performed: In case of a standard GA a single point crossover (called 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 by swapping all characters from position $k + 1$ to l. However, doing so for the above coding scheme could result in infeasible strings. Hence a modification has to be carried out in the standard crossover procedure. To do this 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. The above procedure is repeated till *N* children are obtained.

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.

Plant assignment. Naderi and Ruiz [1] tested several possible rules, from simple to complicated ones, and the NEH2 rule is the best: 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*. We adopt NEH2 to allocate jobs to factories.

Termination criterion. We terminate the GA after 30 generations.

Local search. The local search procedure can be written as follows:

> ***Step 1.*** Specify a seed solution *s*.
>
> ***Step 2.*** Generate a neighborhood set *N*. This is obtained from *s* by interchanging all adjacent pairs of jobs.
>
> ***Step 3.*** Select a schedule *n* in the neighborhood set *N* generated by the seed solution *s*, and compute its fitness value.
>
> ***Step 4.*** If all neighborhood solutions of *s* have been already examined, check the neighborhood solution with the minimum fitness value and improvement ratio. If there is no neighborhood solution that improves the overall solution, terminate this procedure. Otherwise, replace the seed solution with the neighborhood solution with the minimum fitness value and return to Step 3.

Fig.3 Local Search-Based Mutation

Fitness evaluation function. 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 $f_i = r_i / \sum_j r_j$.

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 "Sistem as de Optimización Aplicada SOA" or Applied Optimization Systems (http://soa.iti.es/problem-instances).

In the scheduling area, and more specifically, in the PFSP literature, it is very common to use the benchmark of Taillard [6] that is composed of 12 combinations of $n \times m$: {(20, 50, 100) × (5, 10, 20)}, {200× (10, 20)} and (500×20). Each combination is composed of 10 different replicates so there are 120 instances in total. [1] augment these 120 instances with seven values of F = {2, 3, 4, 5, 6, 7}. This results in a total of 720 large instances. All instances, along with the best known solutions are available at http://soa.iti.es.

We employ the improvement rate 1 (IR1), improvement rate 2 (IR2) , and Gap to describe the effectiveness and efficiency of the proposed hybrid GA. The original makespan is the first of five of the best values of makespan by GA. The GA makespan is the final of five of the best values, the best one of GA. The LS makespan is the proposed local search results. The Best know makespan is from http://soa.iti.es/problem-instances.

$$Improvement\ rate\ I = \left(1 - \frac{GA\ makespan}{orig\text{t}nal\ makespan}\right) \times 100\%$$

$$Improvement\ rate\ II = \left(1 - \frac{LS\ makespan}{GA\ makespan}\right) \times 100\%$$

$$Frequency\ of\ Improvement\ Rate\ II = \frac{Number\ of\ instances\ improved\ by\ LS}{Total\ numer\ of\ instances} \times 100\%$$

$$Gap = \left(1 - \frac{Best\ known\ makespan}{LS\ makespan}\right) \times 100\%$$

Table 1 and Fig. 1 summarize the results of the computational experiments. The improvement Rate I shows that the GA can improve original solution by 1.38-2.63%. The improvement rate II and frequency of improvement rate II shows that the LS algorithm is effective to data with plant number 2 to 5, but not to data with plant number 6 and 7. The average gap shows the efficiency and effectiveness of the proposed hybrid GA. Although the total average gap indicates that the proposed hybrid GA is not powerful enough to improve the currently best known solutions, we hope to design a more powerful one based on the proposed hybrid GA to improve efficiency in the near future.

Conclusions

We propose a hybrid genetic algorithm to solve the distributed permutation flowshop scheduling

problem. In the proposed algorithm we combine a simple genetic algorithm, a local search algorithm and NEH2 to sequencing and assigning jobs to plants. Computational experiments show that the proposed algorithm can improve 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. Combining GA with constraint programming and other heuristics to design hybrid search algorithms will also be a valuable direction.

Table 1: Result summary

Number of plants	2	3	4	5	6	7
Average Improvement Rate I (%)	1.38	1.66	2.16	2.27	2.63	2.22
Average Improvement Rate II (%)	0.06	0.15	0.18	0.15	0.00	0.00
Frequency of Improvement Rate II (%)	11.67	21.67	20.00	18.33	0.00	0.00
Average GAP (%)	17.72	19.68	20.79	21.93	22.78	23.99

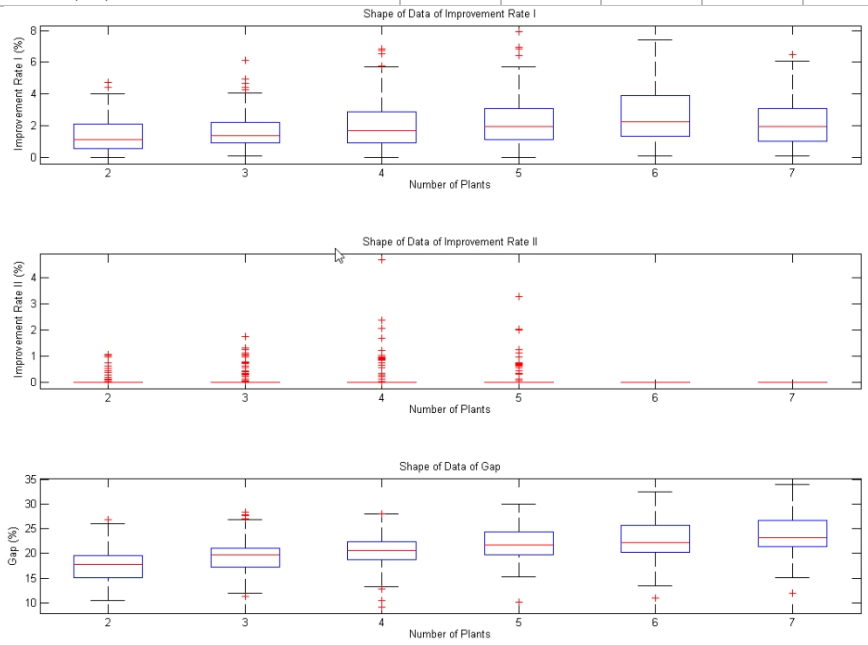

Fig.4. The experimental results

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]Naderi, B. and R. Ruiz, The distributed permutation flowshop scheduling problem. Computers & Operations Research, 2010. 37(4): p. 754-768.

[2]Gao, J. and R. Chen, A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. International Journal of Computational Intelligence Systems, 2011. 4(4): p. 497-508.

[3]Gao, J., R. Chen, and W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. International Journal of Production Research, 2013. 51(3): p. 641-651.

[4]Wang, S.Y., et al., An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. International Journal of Production Economics, 2013.

145(1): p. 387-396.

[5]Lin, S.W., K.C. Ying, and C.Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. International Journal of Production Research, 2013. 51(16): p. 5029-5038.

[6]Taillard, E., Benchmarks for basic scheduling problems. European Journal of Operational Research, 1993. 64(2): p. 278-285.