A Hybrid Genetic Algorithm for the Distributed Permutation Flowshop Scheduling Problem with Sequence-Dependent Setup Times

IOP Conf. Series: Materials Science and Engineering **646** (2019) 012037 doi:10.1088/1757-899X/646/1/012037

A Hybrid Genetic Algorithm for the Distributed Permutation Flowshop Scheduling Problem with Sequence-Dependent Setup Times

Jiangping Huang[1], Quanke Pan[1,*] and Qingda Chen[2]

[1] School of Mechanical and Electrical Engineering and Automation, Shanghai University, Shanghai, China
[2] State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang, China

[*]Email: panquanke@shu.edu.cn

Abstract. The distributed permutation flowshop scheduling problem (DPFSP) has attracted many researchers' attention in recent years. In this paper, we extend the DPFSP by considering the sequence-dependent setup time (SDST). A new hybrid genetic algorithm (HGA) for the DPFSP with the SDST (SDST/DPFSP) is presented to minimize the maximum of the completion time. At first, a new population initialization is proposed. And then, the newly-designed operators are described in details, and we also introduce the mutation rate and the crossover rate to balance the mutation operator and the crossover operator. To further improve the obtained solution, a new local search method is developed. At last, the orthogonal experimental design is applied to adjust the parameters in the HGA, and a comprehensive computational campaign based on the 135 instances demonstrates the effectiveness of the proposed HGA for the SDST/DPFSP.

1. Introduction

The DPFSP, which is a kind of extension of the PFSP, has been an active research field and has impact on many manufacturing industries, such as the compact strip production in the Steel manufacturing industry [1]. In the DPFSP, there exists f identical factories, each of which is a flowshop with m machines and capable of processing all of the jobs independently. The scheduling task of the DPFSP is determining the assignment of the jobs to a certain factory and the processed order of the jobs in a factory. In addition, there are always some operations between the processing of two consecutive jobs such as machine-cleaning, tool-replacing, job-transporting, and so on, which can waste some time called the setup time. Furthermore, the setup time of one job is not only determined by the machine processing it but also by the job processed immediately before it, that is to say, the setup time is sequence-dependent. Therefore, to make the problem studied in the paper more practical, we consider the DPFSP with the SDST to minimize the maximum completion time. To best of our knowledge, there is few literatures of the SDST/DPFSP. Therefore, we will show the related literatures about the DPFSP and the PFSP with the SDST (SDST/PFSP).

The DPFSP is first proposed by Naderi and Ruiz in [2] with the objective of minimizing the makespan. In the following years, many researches have been done to it, and the related effective heuristics and metaheuristics has been developed to address different objectives. Pan, Gao and Wang

present three constructive heuristics and four metaheuristics in [3]. And later, Ruiz, Pan and Naderi propose simple iterated greedy algorithms to minimize the makespan among all factories in [4]. In [5], Pan, Gao and Li address a novel distributed assembly permutation flowshop scheduling problem with three constructive heuristics, two variable neighborhood search methods, and an iterated greedy algorithm. An artificial chemical reaction optimization (CRO), a hybrid discrete cuckoo search (HDCS) algorithm and a new heuristic are proposed in [6][7] and [8] respectively, and the genetic algorithm (GA) is also applied to the DPFSP in [9][10], which has been proved an effective method for the DPFSP. In addition, some other algorithms such as the modified iterated greedy algorithm, the efficient tabu search algorithm are developed in [11][12].

The corresponding literatures of the SDST/PFSP are abundant as well, and we will list as follows. Mirabi [13] presents an ant colony optimization (ACO) algorithm. Rios-Mercado and Bard develop a branch-and-cut algorithm and two different mathematical formulations in [14], and later, they further develop the mentioned algorithms in [15]. Three heuristics, a memetic algorithm, a genetic algorithm and a NEH heuristic are compared in [16] by Kaweegitbundit. Vanchipura and Rajesh develop two constructive heuristic algorithms in [17] and a neighbourhood search known as variable neighbourhood descent (VND) to further improve the two constructive heuristics in [18]. Four iterated local search (ILS) algorithms are constructed in [19] by Wang and Dong. The migrating birds optimization metaheuristic (MBO) is applied to both [20] and [21].

As shown above, the DPFSP and the SDST are both important, and some researches have been done to them respectively, while there is few literature about the SDST/DPFSP. Therefore, we study the SDST/DPFSP in the paper. According to [2], the DPFSP is a NP-Complete problem, and without doubt, the SDST/DPFSP is a NP-Complete problem as well. Therefore, the algorithms applied to the DPFSP are also suitable for the SDST/DPFSP. With the extensive application of the GA, we apply it to the SDST/DPFSP as well. To further improve the quality of the solution we have obtained, we explore the GA with local search, which is called the HGA in the paper.

```
procedure Initialization of one individual
//how to create an individual according to the makespan
  generate a job permutation randomly
  for i := 1 to n do   //where n is the number of the jobs
    calculate the C_max of each factory
    find factory F_min with the minimized C_max
    place job J_i to factory F_min
  endfor
```

Figure 1. The pseudo codes of the initialization of an individual.

```
procedure mutation
  create a random number k
  for i := 1 to k do
    select a factory F_rand randomly
    select a job J_rand randomly from factory F_rand
    replace job J_rand to the best position in all the factories
  endfor
```

Figure 2. The pseudo codes of the mutation operator.

2. Problem description

The SDST/DPFSP can be described as follows. A set of n jobs, $J=\{J_1,J_2,\mathrm{K},J_n\}$, can be processed in f identical factories, $F=\{F_1,F_2,\mathrm{K},F_f\}$, each of which can process one job J_j, ($J_j \in J$) independently just like an independent flowshop, and all of them are with m machines, $M=\{M_1,M_2,\mathrm{K},M_m\}$, respectively. All jobs are processed in the same production routine that starts from the first machine and ends at the last machine with no machine being skipped. The processing time of J_j on machine M_i ($M_i \in M$) is denoted by $p_{j,i}$, the SDST of J_j can be described as $S_{i,j',j}$ on the condition that job J_j is not the first job processed on machine M_i and the job immediately processed before it is job $J_{j'}$ ($J_{j'} \in J$), if the job J_j is the first processed job, the SDST of J_j on machine M_i is denoted as $S_{i,j,j}$. The objective of the SDST/DPFSP in the paper is to find a permutation to minimize the makespan ($C_{\max}$).

3. A hybrid genetic algorithm

In this section, we present the HGA for the SDST/DPFSP with the objective of minimizing the makespan. We will describe the solution representation, population initialization, the design of the operators and the local search method in details and give the pseudo codes.

3.1. Solution representation

We adopt the job-based representation [4], where a solution is represented by a two-dimensional array, $\pi = \{\pi_1, \pi_2, ..., \pi_f\}$. There are f rows, each of which is used to show the permutation of one factory F_l, $\pi_l = (\pi_{l,1}, \pi_{l,2}, ..., \pi_{l,\eta_l})$, $l = 1, 2, ..., f$, where η_l is the total number of jobs assigned to factory F_l. For example, there are two factories and five jobs, and it is assumed that job 1, 2 and 5 are placed in factory F_1, and job 3 and 4 are assigned to factory F_2, as a result, the solution can be represented by $\pi = \{\pi_1, \pi_2\}$, where $\pi_1 = (1, 2, 5)$ and $\pi_2 = (3, 4)$.

```
Procedure Job_Insertion ( )
  set the initial solution π as the reference solution π_ref
  for i := 1 to f do
    for j := 1 to n_i do  // where n_i is the number of
      Job ← π_ref(i, j)   // the jobs processed in factory i
      π ← delete Job inside π
      π ← replace Job to the best position among all factories
    endfor
  endfor
```

Figure 3. The pseudo codes of the Job_Insertion ( )

```
Procedure Job_Exchange ( )
  find factory F_max with the largest makespan
  for i := 1 to n_max do // where n_max is the number of factory F_max
    for j := 1 to f do
      if j ≠ F_max do
        exchange job J_i with each job in factory j and find the best job pairs
      endif
    endfor
  endfor
  exchange the best job pairs
```

Figure 4. The pseudo codes of the Job_Exchange ( )

3.2. Population initialization

It is clear that the quality and the diversity are two important factors for an initial population. To balance the quality and the diversity, we create the individuals in different ways. At first, the NEH2 and the VND(a), both of which are first proposed in [2], are carried out to produce two highly-qualified individuals respectively, while the rest of the individuals are all created according to the makespans of all the factories. In the method, we obtain a random permutation of the jobs at first, and then we reassign the jobs one by one to the factory with the minimized makespan. The pseudo codes of this initialization method are presented in figure 1.

```
procedure crossover
  set the best individual in the population as parent 1
  set a randomly-selected individual in the population as parent 2
  //where parent 1 and parent 2 are different
  for i := 1 to f do   // where f is the number of the factories
    select the random cutting points of each factory for both parent 1 and parent 2
    save the right sides of parent 1 and parent 2 to R2 and R1 respectively
    check parent 1 and R1, and save the remaining jobs to L1 in the order of the appearance
    check parent 2 and R2, and save the remaining jobs to L2 in the order of the appearance
    obtain child 1 by reinserting the jobs in L1 to the best slot of R1
    obtain child 2 by reinserting the jobs in L2 to the best slot of R2
  endfor
```

Figure 5. The pseudo codes of the crossover operator.

```
Procedure Localsearch
  flag ← ture
  π ← perform Job_Exchange ( ) on π
  while flag do
    flag ← false
    π′ ← perform Job_Insertion ( ) on π
    if C_max(π′) < C_max(π) then
      flag ← true
      π ← π′
    endif
  endwhile
```

Figure 6. The pseudo codes of the proposed local search method.

3.3. Design of the operators

In the HGA, there are three operators, the selection operator, the mutation operator and the crossover operator respectively, all of them should be designed carefully since the performance of the algorithm are effected significantly by them. In the following part, the details about the operators are presented. With the selection mechanism, the individuals are ranked according to the fitness and selected for the next iteration. In the paper, the fitness of individuals is equal to $C_{\max}$, which can show the differences among the individuals.

With the mutation operator, the individual should be changed slightly to escape from the local optimization with some good permutation remained. In the mutation operator proposed in the paper, k jobs are selected randomly, where k is a random number which is not more than the half of the total number of jobs. And then the k jobs are reinserted into the best position in all the factories. In this way, on the one hand, it is easy to escape from the local optimization, on the other hand, the new individual can remain the relative positions of the most unchangeable jobs. The pseudo codes of the mutation operator are presented in figure 2.

The crossover operator is applied to two selected individuals for obtaining two new individuals with better fitness. Our crossover operator, for both parents, selects cutting points randomly for all the factories, and the sets of jobs on the right sides of child 1 and child 2 are denoted by $R1$ and $R2$ respectively which are obtained by exchanging the sets of jobs on the right sides of parent 1 and parent 2. The remaining jobs in each sequence of both children are kept in the order of their appearance of their parents, and then they are reinserted to the best position of the right sequence of the corresponding child one by one. In the paper, the two selected individuals are the best one and a randomly-selected one which is different from the best one. The pseudo codes of the crossover operator are presented in figure 5.

Procedure HGA
  set the parameters: *Psize*, $P_{crossover}$, $P_{mutation}$
  while the stopping criterion is not satisfied **do**
    evaluate the fitness of each individual
    apply the local search method on the best individual and a randomly-selected individual
    update the best solution
    apply the selection operator
    make a randomly-selected individual replaced by the best solution
    select another individual randomly that is different from the new best individual
    produce a random number r
    if $r < P_{crossover}$ **then**
      apply crossover operator
    endif
    produce a random number r'
    if $r' < P_{mutation}$ **then**
      apply mutation operator
    endif
  endwhile

Figure 7. The general idea of the HGA.

3.4. Local search method

The local search has been proved an effective method for the PFSP in the related literature. To further improve the solution we have obtained, we present two new local search methods, Job_Insertion ( ) and Job_Exchange ( ), and combine them to the mentioned GA operators. In the Job_Insertion ( ), the reference solution is proposed, at the begin of the Job_Insertion ( ), we set the initial solution as the reference solution, and test the jobs in the order of the reference solution one by one to find the best position and place them to it. The pseudo codes of the Job_Insertion ( ) are presented in figure 3. In the Job_Exchange ( ), at first, we should find the factory $F_{\max}$ with the maximum makespan among the

factories, and then exchange the jobs in factory $F_{\max}$ one by one with the jobs in the other factories to find the best job pairs (one job from factory $F_{\max}$, and the other one from one of the other factories), and exchange the positions between them. The pseudo codes of the Job_Exchange() are presented in figure 4. In the method of the proposed local search, the Job_Exchange() operator is used to the individual at first. And then the Job_Insertion() is applied to the new obtained individual repeatedly until no improvement can be obtained. Figure 6 presents the pseudo codes of the proposed local search method.

3.5. An overview of the HGA

The general idea of the HGA is presented in figure 7. At first, the parameters, *Psize*, $P_{crossover}$, $P_{mutation}$, should be set, where *Psize* is the size of the population, $P_{crossover}$ and $P_{mutation}$ are the crossover rate and the mutation rate respectively. Secondly, the fitness of each individual should be calculated which is used to rank the individuals. Thirdly, the local search method is applied to the best individual and a randomly-selected individual, then the best solution should be updated which is used to replace a randomly-selected individual different from the best one. Fourthly, besides the new best individual, another individual should be selected randomly that is for the following crossover operation and mutation operation. At last, we create two random numbers r and r' for the crossover operation and the mutation operation respectively. While r is less than $P_{crossover}$, the crossover operator should be employed to the best individual and the randomly-selected individual, and if r' is less than $P_{mutation}$, both the individuals are operated by the mutation operator.

4. Computational Experiment

The 135 problem instances used to test the performance of the algorithms are listed as follows. The combinations of $n \times m \times f$ are: $\{100, 200, 300, 400, 500\} \times \{5, 8, 10\} \times \{2, 3, 4\}$, and the setup times are generated by $\text{Setuptime} = (1 + rand(\,)\%99) \times \text{Factor} / 100$, where Factor is selected from $\{25, 50, 100\}$ randomly, and $rand(\,)$ is a random number. All of the algorithms are coded in C++ and compiled with Visual Studio 2013, and the algorithms share most important functions in the code. All the experiments in this paper are performed on windows 10 machine with 1 processor, i3-8100, and 8GB of RAM memory.

Table 1. RPI values for $t = 20 \cdot m \cdot n$ Milliseconds

Instance Size	$f = 2$		$f = 3$		$f = 4$	
	HGA(1)	HGA	HGA(1)	HGA	HGA(1)	HGA
100×5	15.31	**2.50**	14.22	**2.42**	15.10	**2.57**
100×8	14.84	**1.88**	16.03	**2.5**	16.15	**2.71**
100×10	15.21	**2.57**	15.50	**2.09**	14.70	**2.64**
200×5	11.65	**1.86**	12.99	**1.42**	12.21	**1.98**
200×8	12.46	**1.35**	12.78	**1.19**	14.50	**2.10**
200×10	12.57	**2.36**	12.01	**1.81**	13.32	**2.44**
300×5	11.26	**1.24**	12.01	**1.38**	13.52	**2.31**
300×8	10.51	**1.49**	11.26	**2.03**	12.97	**1.39**
300×10	10.54	**1.20**	11.76	**1.55**	9.57	**1.80**
400×5	10.86	**2.37**	9.51	**1.24**	11.04	**1.64**
400×8	9.57	**1.70**	11.54	**1.57**	11.27	**1.26**
400×10	9.59	**2.25**	10.43	**1.58**	9.87	**1.36**
500×5	9.73	**4.04**	10.25	**2.42**	10.70	**3.12**
500×8	9.39	**3.72**	9.06	**2.70**	10.24	**3.96**
500×10	9.77	**2.94**	9.79	**2.84**	10.07	**2.79**
mean	11.55	**2.23**	11.94	**1.92**	12.35	**2.27**

In this paper, we apply the termination criterion with a predefined CPU time, $t = C \cdot m \cdot n$, the algorithm has elapsed, where C is a predefined value. The 3 parameters, *Psize*, $P_{crossover}$, $P_{mutation}$, are fixed as follows: *Psize*=40, $P_{crossover}$=0.1, $P_{mutation}$=0.1, and all of them are obtained by the Orthogonal experimental design with an orthogonal array $L_{16}(4^3)$ within $10 \cdot m \cdot n$ milliseconds.

We compare the presented HGA with the HGA(1) proposed in [12]. In the paper, both the HGA(1) and the HGA are tested to solve the 135 instances mentioned in this section with 5 replications, and they are also terminated with the same CPU time where the value of C has been tested at 3 levels, 10, 20 and 30 respectively in the paper. The HGA shows better performance among the experiments with the three levels. Due to the limited space of the paper, we just list the RPI values of C=20 in table 1. From table 1, we can see that the means of the RPI generated by the presented HGA are much smaller than those of the HGA(1) nevertheless the different factories involved.

5. Conclusion

In the paper, we address the SDST/DPFSP with minimization of the makespan. We propose a HGA for such a problem, and the results of the numerical experiments show that the developed algorithm has a better performance than the existing HGA(1). In the future, we will further improve the HGA by developing more effective local search methods and the operators, and more related algorithms should be compared together.

Acknowledgments

This research is partially supported by the National Science Foundation of China 51575212 and 61174187.

References

[1] Quan-Ke Pan, Liang Gao, Ling Wang, 2019, *Applied Mathematical Modelling*, Volume 73, Pages 327-348
[2] B. Naderi, Ruben Ruiz, 2010, *Computers & Operations Research*, Volume 37, Issue 4, 754-768
[3] Quan-Ke Pan, Liang Gao, Ling Wang, Jing Liang, Xin-Yu Li, 2019, *Expert Systems with Applications*, Volume 124, 309-324
[4] Ruben Ruiz, Quan-Ke Pan, Bahman Naderi, 2019, *Omega*, Volume 83, 213-222
[5] Quan-Ke Pan, Liang Gao, Li Xin-Yu, Framinan M. Jose, 2019, *Applied Soft Computing*,, Volume 81, 105492
[6] Hafewa Bargaoui, Olfa Belkahla Driss, Khaled Ghedira, 2017, *Computers & Industrial Engineering*, Volume 111, 239-250
[7] J. Wang, L. Wang and J. Shen, "*A hybrid discrete cuckoo search for distributed permutation flowshop scheduling problem*," 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, 2016, 2240-2246.
[8] Fernandez-Viagas V, Framinan JM, 2015, *International Journal Of Production Research*, Vol. 53, No. 4, 1111–112m
[9] Yan Li, Zhigang Chen, Yan Li, "*The distributed permutation flowshop scheduling problem: A genetic algorithm approach*," 2015, ICMII 2015, Atlantis Press, 2352-538X
[10] J. Wang, L. Wang and J. Shen, "*A hybrid discrete cuckoo search for distributed permutation flowshop scheduling problem*," 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, 2016, 2240-2246.
[11] Lin Shih-Wei, Ying Kuo-Ching, Huang Chien-Yi, 2013, *International Journal of Production Research*, Volume 51, Issue 16, 5029-5038
[12] Gao Jian, Chen Rong, Deng Wu, 2013, *International Journal of Production Research*, Volume 51, Issue 3, 641-651
[13] Mohammad Mirabi, 2010, *The International Journal of Advanced Manufacturing Technology*, Volume 55, Issue 1-4, 317-326

[14] Roger Z. Rios-Mercado, Jonathan F. Bard, 1998, *Computers & Operations Research*, Volume 25, Issue 5, 351-366
[15] Roger Z Rios-Mercado, Jonathan F Bard, 2003, *Journal of Combinatorial Optimization*, Volume 7, Issue 3, 291-318
[16] P. Kaweegitbundit, 2011, *Advanced Materials Research*, Volume 339, 332-335
[17] Vanchipura R, Sridharan R, 2013, *The International Journal of Advanced Manufacturing Technology*, Volume 67, Issue 5-8, 1337-1353
[18] Rajesh Vanchipura, R. Sridharan, A. Subash Babu, 2014, *Journal of Manufacturing Systems*, Volume 33, Issue 1, 65-75
[19] Wang Y, Dong X, Chen P, Lin Y, 2014, *Advances in Intelligent Systems and Computing*, vol 277, 329-338
[20] Imene Benkalai, Djamal Rebaine, Caroline Gagné, Pierre Baptiste, 2016, *IFAC-PapersOnLine*, Volume 49, Issue 12, 408-413
[21] A. Sioud and C. Gagne, "*An MBO algorithm for a flow shop problem with sequence-dependent setup times*," 2016 12th World Congress on Intelligent Control and Automation (WCICA), Guilin, 2016, pp. 2471-2474.