Int j simul model 18 (2019) 2, 335-343
ISSN 1726-4529 Original scientific paper

A SOLUTION TO SINGLE-MACHINE INVERSE JOB-SHOP SCHEDULING PROBLEM

Wang, Y.[*,#]; Yang, O.[**] & Wang, S. N.[*]
[*] School of Electronic and Communication Engineering, Shenzhen Polytechnic, Shenzhen 518055, China
[**] School of Computer Engineering, Shenzhen Polytechnic, Shenzhen 518055, China
E-Mail: wyang@szpt.edu.cn ([#] Corresponding author)

Abstract
Concerning the inverse job-shop scheduling problem (JSP), this paper proposes a hybrid solution based on genetic algorithm (GA) and improved particle swarm optimization (PSO), with the aim to minimize the parameter adjustment. The solution was presented as a block coding plan with decimal mechanism, under which both processes and parameters can be optimized simultaneously. To enhance the local search ability of the proposed algorithm, four neighbourhood structures were designed, and an adaptive selection mechanism was created to select the most suitable neighbourhood. Finally, the proposed algorithm was proved valid through discrete event simulation (DES) and comparison with other algorithms.
(Received, processed and accepted by the Chinese Representative Office.)

Key Words: Inverse Scheduling, Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Job-Shop Scheduling Problem (JSP), Discrete Event Simulation (DES)

1. INTRODUCTION

Job-shop scheduling problem (JSP) is indispensable to production management of enterprises. It aims to meet or optimize one or more objectives by allocating limited resources to several tasks in time [1, 2]. In traditional JSPs [3, 4], the scheduling optimization is often carried out under the assumption that both the processing information of the jobs and the job-shop conditions are known in advance and not time variant. In actual production, however, the job-shop conditions have dynamic changes, calling for a scheduling method with high flexibility and fault tolerance.

In recent years, the inverse optimization [5], a theory first proposed in 1992, has been introduced to solve the JSP, giving birth to inverse scheduling algorithms. The inverse scheduling attempts to optimize feasible scheduling plans by minimizing the estimated value of processing parameters and identifying the variation of the minimal value [6]. Currently, inverse scheduling is mostly solved by accurate algorithms like linear programming, which cannot be applied to largescale or complex JSPs.

In light of the above, a hybrid solution to the inverse JSP was designed from genetic algorithm (GA) and improved particle swarm optimization (PSO), with the aim to minimize the parameter adjustment. The solution was presented as a block coding plan with decimal mechanism, under which both processes and parameters can be optimized simultaneously. To enhance the local search ability of the proposed algorithm, four neighbourhood structures were designed, and an adaptive selection mechanism was created to select the most suitable neighbourhood. Finally, the proposed algorithm was proved valid through discrete event simulation (DES) and comparison with other algorithms.

2. LITERATURE REVIEW

The scheduling objective varies from job-shop to job-shop. With the globalization of economy, job-shop production now needs to satisfy the growing demand for various products

in small batches. As a result, many new JSP methods have emerged, such as multi-objective flexible JSP [7], hybrid JSP under lean production [8], reconfigurable JSP [9] and flexible, low-carbon, energy-saving JSP [10].

In real-world JSPs, it is often possible to find a feasible solution according to a set of known parameters. However, the feasible solution is not necessarily the optimal one. To optimize the solution, the objective should be achieved with the minimal number of parameters. The minimization process is called the inverse optimization [11]. With the introduction of the inverse optimization, the JSP can be transformed into the inverse JSP, which is NP-hard. Brucker and Shakhlevich [12] were the first to discuss the inverse JSP, or more specifically, the single-machine inverse JSP. In their research, the inverse JSP was solved with two parameters (i.e. makespan and delivery time) under five distances, with the goal to minimize the maximum tardiness, and proved NP-hard in all scenarios. Koulamas [13] explored the inverse JSP with controllable makespan based on Johnson's rule. Mou et al. [14] studied the modelling and solution of inverse JSP, converted the problem into different mathematical forms, and presented a solution with no weight and minimal makespan. Peng et al. [15] attempted to minimize the total weighted makespan of the inverse JSP for parallel machines, and obtained the optimal solution by linear programming and quadratic programming.

Due to its NP-hardness, not every inverse JSP has been modelled. The existing models are limited to single-machine JSP and two-machine JSP. What is worse, most of the existing models only consider a few factors, such as the variable makespan, failing to take account of the various changing processing parameters in real life. In addition, the inverse JSP is usually solved by simple linear programming and quadratic programming. The solving methods only apply to simple, small-scale scenarios. Considering the existing models and the features of various JSPs, this paper attempts to design an efficient hybrid solution to inverse JSP based on the GA and the PSO.

3. PROBLEM MODELLING

3.1 Introduction to single-machine JSP

Single-machine JSP is the basic form and a special case of production scheduling. The solution to this problem provides reference for other complex JSPs. In fact, a complex JSP in actual production can be decomposed into several single-machine JSPs. According to the theory of permutations and combinations, there are $n!$ permutations in a n-job single-machine JSP. Obviously, the solution to such a problem requires tedious computations. If n is sufficiently large, the single-machine JSP can be regarded as NP-hard defying the traditional precise algorithms.

Traditionally, the single-machine JSP regards processing information as determinate parameters. As a result, the processing parameters and optimization targets are both considered as constant in the scheduling plan prepared by the known information. When applied in actual JSPs, the scheduling plan often loses its optimality due to the changing states of the job-shop, and even becomes infeasible. Meanwhile, the scheduling sequence cannot be adjusted easily or fine-tuned before the production completes. Therefore, it is imperative to adjust the relevant parameter, such that the scheduling plan can achieves the objective at the minimal cost or with the minimal changes. This is also the essence of single-machine reverse JSP.

3.2 Modelling of single-machine inverse JSP

Let n be the number of jobs to be processed on the single-machine job-shop. The n jobs have different processing parameters, such as delivery time, processing time and weight. These

parameters have been estimated before production and can be adjusted in a reasonable range. The scheduling plan, which is prepared based on the estimated parameters, aims to achieve the scheduling objective at the minimal cost or with the minimal changes. The following assumptions were put forward for the modelling of the single-machine inverse JSP:

- The machine is idle at the start and available at any time.
- There is no preemption of job in the production process.
- The machine has no constraint on caching.
- The machine can process one job only at a time.
- The preparation time is negligible.
- There is no constraint on the processing order.
- The processing parameters are adjusted in a reasonable range.

Based on the above assumptions, a single-machine inverse JSP model can be established to minimize the total weighted makespan.

In this problem, the relevant parameters can be divided into the initial group and the adjusted group. The former includes the initial processing time t_j, initial job weight w_j and initial job makespan c_j, while the latter consists of the adjusted processing time $\hat{t}_j$, adjusted job weight $\hat{w}_j$ and adjusted job makespan $\hat{c}_j$. The objective of minimizing the total weighted makespan is equivalent to optimizing the original scheduling plan τ under $\hat{t}_j$ without changing w_j and adjusted t_j. The processing time of each job t_j can be adjusted in the interval $[\underline{t}_j, \hat{t}_j]$. The minimum parameter change is denoted as X. Hence, the corresponding single-machine inverse JSP model can be expressed as follows.

Objective function:

$$\min X = \min \sum \| t_j - \hat{t}_j \| \tag{1}$$

Constraints:

$$\frac{\hat{t}_1}{w_1} \leq \frac{\hat{t}_2}{w_2} \leq \cdots \leq \frac{\hat{t}_n}{w_n} \tag{2}$$

$$w_1\hat{t}_1 + w_2(\hat{t}_1 + \hat{t}_2) + \cdots + w_n \sum_{j=1}^{n} \hat{t}_j \leq \sum_{j=1}^{n} w_j c_j \tag{3}$$

$$\hat{t}_j \geq 0, j = 1,2,\ldots,\mathrm{n} \tag{4}$$

where, constraint (2) guarantees the optimality of the original scheduling plan τ under $\hat{t}_j$; constraint (3) ensures that the adjusted objective value, i.e. the total weighted makespan, is smaller than the original objective value; constraint (4) assures that the total processing time of all jobs is not negative.

4. INVERSE SCHEDULING ALGORITHM BASED ON GA AND PSO

GA [16-20] is a stochastic search algorithm for optimal solution. In the search process, each population is selected according to individual fitness, and subjected to crossover and mutation, creating the next generation of population. The new population is more adaptable than the parent population. Then, the optimal individuals in the population are decoded to obtain an approximate optimal solution.

4.1 Chromosome operations

(1) Coding

Chromosome coding, a key step of the GA, completes the mapping from the solution space to the coding space. The general JSPs are often treated by process-based coding. On the one hand, the jobs to be processed are numbered $1, 2, \ldots, n$ in the order of processing; on the

other hand, a sequence of chromosomes is generated by random as per the process relationship, with each chromosome containing the information of a scheduling plan. Nevertheless, the process-based coding forbids the adjustment of processing parameters during the operation, which is required for single-machine inverse JSP. In this paper, the processing time is adjusted in a controllable range by a real-number coding plan that makes the algorithm feasible and operator operable. The details of the coding plan are shown in Fig. 1 below.

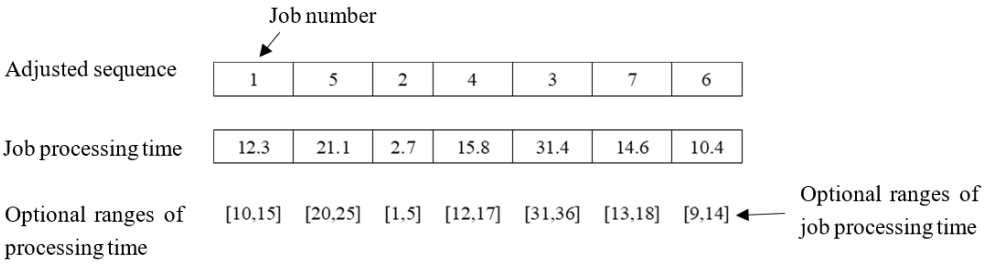

Figure 1: Coding plan for single-machine inverse JSP.

(2) Initialization

Population initialization directly affects the performance of the evolutionary algorithm. For example, the quality of initial solution may bear on the efficiency of the GA. Since the inverse JSP needs to adjust a given nonoptimal scheduling plan, an initial nonoptimal scheduling sequence is needed before the execution of the algorithm.

In this paper, the heuristic nonoptimal scheduling is combined with random initialization and local initialization to generate the initial population for inverse JSP. The combined method can enhance the diversity of chromosomes and the quality of individuals in the initial population.

With the minimal total weighted makespan as the objective, our single-machine inverse JSP can be denoted as $1||\sum w_j c_j$. In traditional scheduling, the optimal solution to the $1||\sum w_j c_j$ problem must satisfy $\frac{\hat{t}_1}{w_1} \leq \frac{\hat{t}_2}{w_2} \leq \ldots \leq \frac{\hat{t}_n}{w_n}$. The first step of initialization is the initial sorting operation. Here, the initial optimal sequence for inverse JSP is obtained under the weighted shortest processing time (WSPT) rules [21], according to the original processing information or estimated parameter information. Several jobs were swapped randomly to make the optimal sequence nonoptimal, creating an inverse scheduling environment.

The adopted initialization method consists of the following steps:

Step 1: The optimal scheduling sequence is determined under the WSPT rules according to the original processing information.

Step 2: The positions of several jobs are swapped randomly to make the optimal sequence nonoptimal, creating an inverse scheduling environment.

Step 3: *M* chromosomes are generated by random initialization at the probability of *R*.

Step 4: (*N*-*M*) chromosomes are generated by local initialization, and mixed into the initial population for inverse JSP.

(3) Crossover

Crossover is an operation to produce new individuals for the population in the next generation. The operation is realized by gene recombination. The effective pattern of genes is preserved as far as possible, such as to enhance the search ability of the algorithm. Considering the features of inverse JSP, uniform crossover was adopted for our population. The specific steps are as follows:

Step 1: A random integer *r* is selected from the interval [1, T_0].

Step 2: *r* irrelevant integers are generated.

Step 3: The genes at the r^{th} position in two parent chromosomes, P1 and P2, are reserved in offspring chromosomes, O1 and O2, respectively, and the other genes in the parent population are mapped randomly to the offspring population.

Fig. 2 gives an example of uniform crossover with $r = 4$. The four randomly generated integers are 1, 3, 6 and 7. Thus, the genes at the 1^{st}, 3^{rd}, 6^{th} and 7^{th} positions were directly copied to the same positions in the offspring population, while the genes at the 2^{nd}, 4^{th} and 5^{th} positions were subjected to cross reproduction.

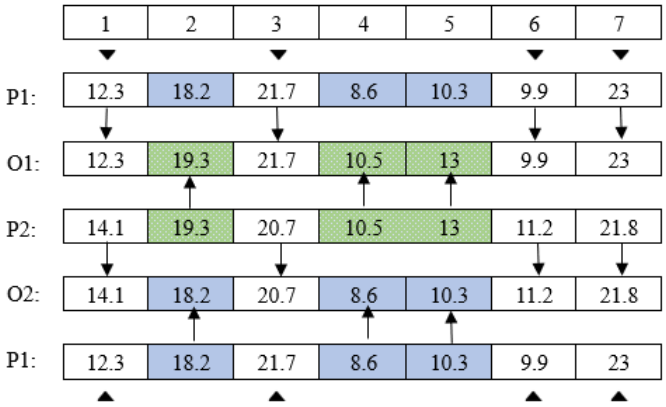

Figure 2: An example of uniform crossover.

(4) Mutation

In the mutation operation, new individuals are generated by replacing alleles in the GA, with the aim to maintain population diversity, enhance local search ability and prevent premature convergence. In this paper, the single-machine inverse JSP is solved by swap mutation through the following steps:

Step 1: Two positions are selected randomly from the parent chromosome.

Step 2: The genes at the two positions are swapped to create the offspring chromosome.

Fig. 3 offers an example of the swap mutation. It can be seen that the 2^{nd} and 4^{th} genes of the parent chromosome P1 were selected and exchanged, creating the offspring chromosome O1.

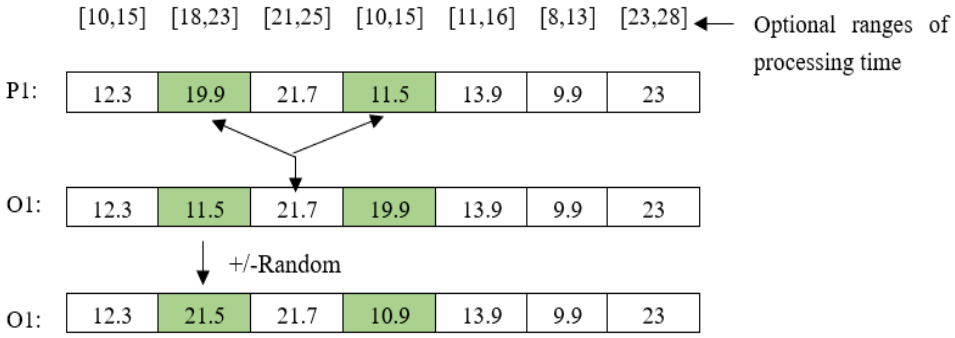

Figure 3: An example of swap mutation.

4.2 Solution to inverse JSP based on GA and improved PSO

Our hybrid solution to inverse JSP was designed from GA and improved PSO. The two algorithms were integrated into a parallel mechanism that gives full play to their respective advantages, making it possible to enhance local search ability and accelerate the convergence. The basic flow of the hybrid solution is detailed in Fig. 4 below.

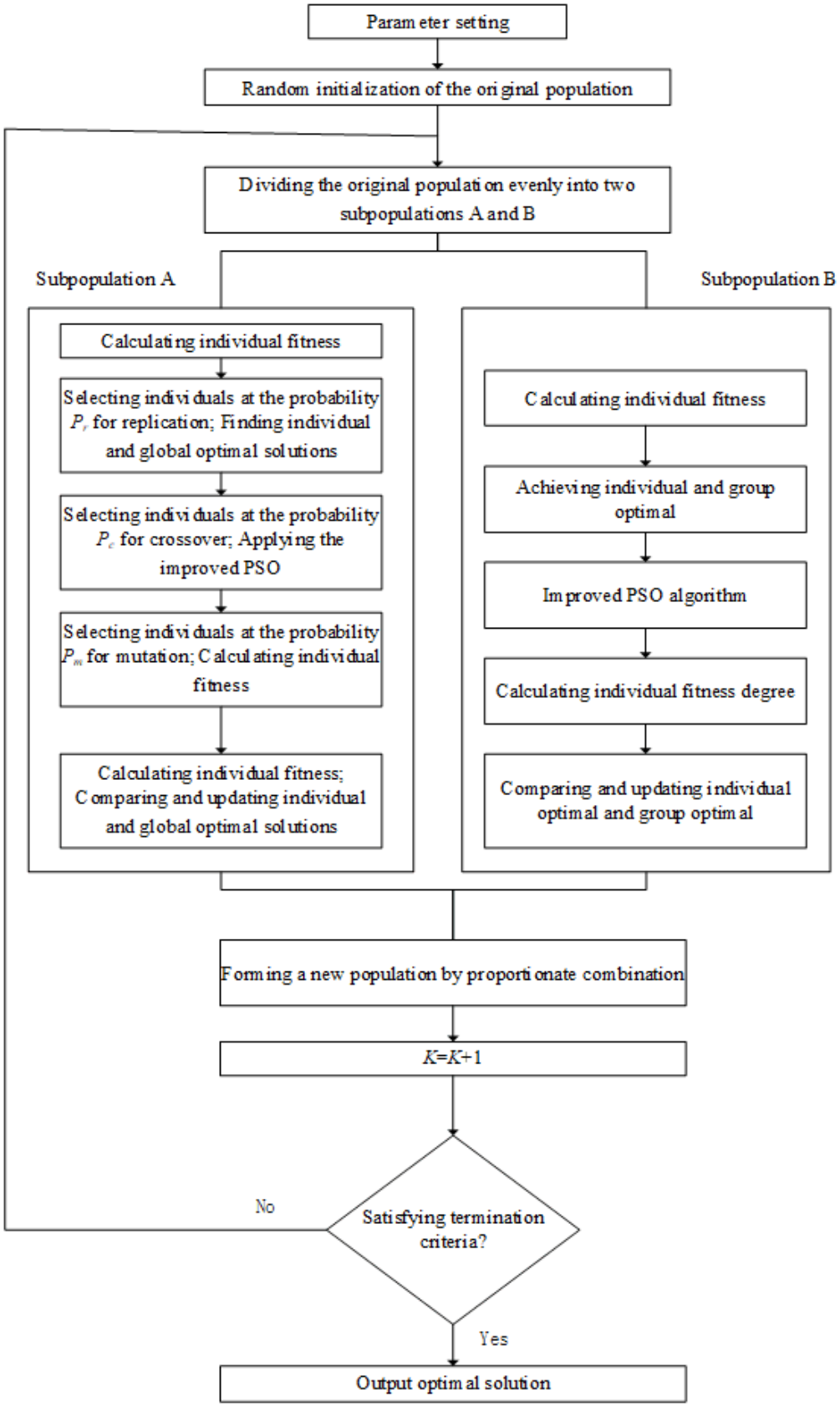

Figure 4: The basic flow of the hybrid solution.

The solving steps are explained below.

Step 1: The parameters of the hybrid solution are configured, including population size N, number of iteration T, random initialization ratio R_r, number of parameter changes in original

chromosomes N_c, crossover probability P_c, mutation probability P_m, replication probability P_r, etc.

Step 2: Nonoptimal scheduling sequences are generated under the WSPT rules and the proposed initialization method. The heuristic nonoptimal scheduling is combined with random initialization and local initialization by the ratio of R_r to N_c, and used to generate an initial population with good inverse scheduling quality.

Step 3: The population is divided into two subpopulations A and B, which are respectively treated with the GA and the improved PSO.

Step 4: The fitness of individual chromosomes in the population is evaluated. Then, a new population is generated with 25 % of its individuals from subpopulation A, 25 % from subpopulation B, 30 % from the mixed population, and 20 % from the original population. Note that those from subpopulation A are the best individuals after GA operation; those from subpopulation B are the best individuals after improved PSO operation; those from the mixed population are determined after evolution; and those from the original population are selected by random. To prevent duplication, a new evolutionary population is formed and subjected to the next iteration.

Step 5: Let $K = K + 1$. Then, the result is outputted if the termination criteria are satisfied, or the process is repeated from Step 3.

5. SIMULATION AND ANALYSIS

This section verifies our hybrid solution through simulation experiments on experiments of different scales. The parameters of the three examples are as follows: the number of jobs is 20, 50 and 100, respectively; the interval of processing time p, which obeys uniform distribution, is [5, 25], [30, 100] and [80, 120], respectively; the interval of the weight of job w, which obeys discrete uniform distribution, is [10, 50], [10, 100] and [100, 200], respectively. To simulate the parameter initialization of our method, the three parameters were combined in different patterns, forming 27 instance sets. The parameter setting is specified in Table I below.

Table I: Parameter setting.

Simulation parameters	Scale	Parameter value
Number of jobs (n)	Small scale	20
	Medium scale	50
	Large scale	100
Processing time (p)	Small scale	[5, 25]
	Medium scale	[30, 100]
	Large scale	[80, 120]
Job weight (w)	Small scale	[10, 50]
	Medium scale	[10, 100]
	Large scale	[100, 200]

The parameters of our hybrid solution were partially obtained empirically and partially through experiments. The solution parameters were eventually configured as: population size $N = 100$, the maximum number of iterations $T = 300$, the replication probability $P_r = 0.02$, the crossover probability $P_c = 0.75$, the mutation probability $P_m = 0.12$, the job weight $w = 0.9$, and constants $c_1 = c_2 = 2$.

The effect of our initialization method hinges on two parameters: random initialization ratio R_r and number of parameter changes in original chromosomes N_c. As mentioned before, the random initialization and local initialization were combined to ensure population diversity. In local initialization, some job parameters are assumed as constants and others are

generated randomly. To verify the initialization effect, the value of R_r was set to 0.3, 0.6 and 0.9, respectively, and N_c to 25 %, 50 % and 75 %, respectively. In this way, nine different instances were set up for simulation.

To disclose the effect of different parameters on the hybrid solution, the experiments were carried out in the same environment, each instance was tested 20 times in a row, and the mean result was obtained for statistical analysis. The experimental results are listed in Table II.

Table II. Experimental results of different instances.

Parameter combination	20 jobs		50 jobs		100 jobs	
	RPD	*Z*	*RPD*	*Z*	*RPD*	*Z*
(0.3, 25 %)	0.13	12	1.62	50	0.72	75
(0.3, 50 %)	0.45	12	0.27	46	1.18	78
(0.3, 75 %)	1.12	18	0.21	55	0.49	77
(0.6, 25 %)	2.18	22	0.13	40	1.01	81
(0.6, 50 %)	0.56	25	1.81	64	1.81	76
(0.6, 75 %)	0.61	31	0.25	46	0.22	68
(0.9, 25 %)	1.58	26	1.10	56	0.66	80
(0.9, 50 %)	0.58	23	0.39	52	1.20	82
(0.9, 75 %)	0.62	24	0.87	50	1.07	84

RPD – Residual prediction deviation; *Z* – Standardized statistic

As shown in Table II, different parameter combinations brought different simulation results for each instance, whichever the scale of the problem. For small scale problems (20 jobs), the optimal results were achieved at $R_r = 0.3$ and $N_c = 25$ %. For medium scale problems (50 jobs), the optimal results were realized at $R_r = 0.6$ and $N_c = 25$ %. For large scale problems (100 jobs), the optimal results were obtained at $R_r = 0.3$ and $N_c = 75$ %.

6. CONCLUSIONS

This paper models the inverse JSP with minimal total weighted makespan, and proposes a hybrid solution based on GA and improved PSO to solve the model. In the hybrid solution, the heuristic nonoptimal scheduling is combined with random initialization and local initialization to produce the initial population for inverse JSP. The combined approach enhances the population diversity and quality. Considering the problem features and existing coding methods, effective crossover and mutation operators were designed to improve the PSO. Finally, the hybrid solution was verified by DES, as well as the coding method and operators.

ACKNOWLEDGEMENT

This paper was supported by Guangdong IIOT(M-S) Engineering Technology Center (No.2015-1487), Guangdong IIOT Engineering Laboratory (No.2018-3149) and Shenzhen IIOT Engineering Laboratory (No.2017-823).

REFERENCES

[1] Spanos, A. C.; Ponis, S. T.; Tatsiopoulos, I. P.; Christou, I. T.; Rokou, E. (2014). A new hybrid parallel genetic algorithm for the job-shop scheduling problem, *International Transactions in Operational Research*, Vol. 21, No. 3, 479-499, doi:10.1111/itor.12056

[2] Chen, Q.; Deng, L. F.; Wang, H. M. (2018). Optimization of multi-task job-shop scheduling based on uncertainty theory algorithm, *International Journal of Simulation Modelling*, Vol. 17, No. 3, 543-552, doi:10.2507/IJSIMM17(3)CO14

[3] Yusof, R.; Khalid, M.; Hui, G. T.; Yusof, S. M.; Othman, M. F. (2011). Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm, *Applied Soft Computing*, Vol. 11, No. 8, 5782-5792, doi:10.1016/j.asoc.2011.01.046

[4] Chang, H.-C.; Chen, Y.-P.; Liu, T.-K.; Chou, J.-H. (2015). Solving the flexible job shop scheduling problem with makespan optimization by using a hybrid Taguchi-genetic algorithm, *IEEE Access*, Vol. 3, 1740-1754, doi:10.1109/ACCESS.2015.2481463

[5] Burton, D.; Toint, P. L. (1992). On an instance of the inverse shortest paths problem, *Mathematical Programming*, Vol. 53, No. 1-3, 45-61, doi:10.1007/BF01585693

[6] Mou, J.; Liang, G.; Guo, Q.; Mu, J. (2017). A hybrid heuristic algorithm for flowshop inverse scheduling problem under a dynamic environment, *Cluster Computing*, Vol. 20, No. 1, 439-453, doi:10.1007/s10586-017-0734-6

[7] Shao, X.; Liu, W.; Liu, Q.; Zhang, C. (2013) Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem, *International Journal of Advanced Manufacturing Technology*, Vol. 67, No. 9-12, 2885-2901, doi:10.1007/s00170-012-4701-3

[8] Dhingra, A.; Chandna, P. (2010). Multi-objective flow shop scheduling using hybrid simulated annealing, *Measuring Business Excellence*, Vol. 14, No. 3, 30-41, doi:10.1108/13683041011074191

[9] Azab, A.; Naderi, B. (2015). Modelling the problem of production scheduling for reconfigurable manufacturing systems, *Procedia CIRP*, Vol. 33, 76-80, doi:10.1016/j.procir.2015.06.015

[10] Jiang, T.; Deng, G. (2018). Optimizing the low-carbon flexible job shop scheduling problem considering energy consumption, *IEEE Access*, Vol. 6, 46346-46355, doi:10.1109/ACCESS.2018.2866133

[11] Sun, G.; Bin, S. (2017). Router-level internet topology evolution model based on multi-subnet composited complex network model, *Journal of Internet Technology*, Vol. 18, No. 6, 1275-1283, doi:10.6138/JIT.2017.18.6.20140617

[12] Brucker, P.; Shakhlevich, N. V. (2009). Inverse scheduling with maximum lateness objective, *Journal of Scheduling*, Vol. 12, No. 5, 475-488, doi:10.1007/s10951-009-0117-9

[13] Koulamas, C. (2005). Inverse scheduling with controllable job parameters, *International Journal of Services and Operations Management*, Vol. 1, No. 1, 35-43, doi:10.1504/IJSOM.2005.006316

[14] Mou, J.; Gao, L.; Guo, Q.; Xu, R.; Li, X. (2018). Hybrid optimization algorithms by various structures for a real-world inverse scheduling problem with uncertain due-dates under single-machine shop systems, *Neural Computing and Applications*, 18 pages (in press), doi:10.1007/s00521-018-3472-7

[15] Guo, P.; Wenming, C.; Wang, Y. (2015). Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm, *Engineering Optimization*, Vol. 47, No. 11, 1564-1585, doi:10.1080/0305215x.2014.982634

[16] Sun, G.; Bin, S. (2018). A new opinion leaders detecting algorithm in multi-relationship online social networks, *Multimedia Tools and Applications*, Vol. 77, No. 4, 4295-4307, doi:10.1007/s11042-017-4766-y

[17] Gopi, A. P.; Narayana, V. L.; Kumar, N. A. (2018). Dynamic load balancing for client server assignment in distributed system using genetic algorithm, *Ingénierie des Systèmes d'Information*, Vol. 23, No. 6, 87-98, doi:10.3166/ISI.23.6.87-98

[18] Moslemi, H. R.; Keshtkar, M. M. (2018). Sensitivity analysis and thermal performance optimization of evacuated U-tube solar collector using genetic algorithm, *International Journal of Heat and Technology*, Vol. 36, No. 4, 1193-1202, doi:10.18280/ijht.360406

[19] Jemilda, G.; Baulkani, S. (2018). Moving object detection and tracking using genetic algorithm enabled extreme learning machine, *International Journal of Computers Communications and Control*, Vol. 13, No. 2, 162-174, doi:10.15837/ijccc.2018.2.3064

[20] Ramanauskas, M.; Šešok, D.; Belevičius, R.; Kurilovas, E.; Valentinavičius, S. (2017). Genetic algorithm with modified crossover for grillage optimization, *International Journal of Computers Communications and Control*, Vol. 12, No. 3, 393-401, doi:10.15837/ijccc.2017.3.2813

[21] Kacem, I.; Chu, C. (2008). Worst-case analysis of the WSPT and MWSPT rules for single machine scheduling with one planned setup period, *European Journal of Operational Research*, Vol. 187, No. 3, 1080-1089, doi:10.1016/j.ejor.2006.06.062