Hindawi
Discrete Dynamics in Nature and Society
Volume 2021, Article ID 8166343, 10 pages
https://doi.org/10.1155/2021/8166343

Research Article

An Automatic Heuristic Design Approach for Seru Scheduling Problem with Resource Conflicts

Rongxin Zhan[1], **Jinhui Zhang**[2], **Zihua Cui**[3], **Jin Peng**[4] and **Dongni Li**[1,5]

[1]*School of Computer Science, Beijing Institute of Technology, Beijing, China*
[2]*School of Automation, Beijing Institute of Technology, Beijing, China*
[3]*Guangdong Shenling Environmental Systems Co., Ltd., Foshan, China*
[4]*China Nuclear Power Engineering Co., Ltd., Shenzhen, China*
[5]*Southeast Academy of Information Technology, Beijing Institute of Technology, Putian, China*

Correspondence should be addressed to Jin Peng; peng.jin@cgnpc.com.cn

Received 30 July 2021; Accepted 2 December 2021; Published 21 December 2021

Academic Editor: Shi Cheng

In current environments, production systems need the ability of quick response to face the volatile markets. Seru production systems, as a new mode of the production system, have the advantages of quick response, high flexibility, and high efficiency. Seru scheduling, which refers to constructing serus with an exact sequence in limited workspace, is an important decision problem in the operational management of seru production systems and can reflect the reconfiguration nature of seru production systems. This study investigates a seru scheduling problem with resource conflicts, whose objective is to minimize the makespan. An automatic heuristic design approach that combines a genetic programming algorithm and structure similarity-based operators is proposed. Comparative experiments are conducted with human-made rules, basic genetic programming, genetic programming-based algorithm, and with some state-of-the-art scheduling algorithms. The results show the effectiveness and efficiency of the proposed algorithm.

1. Introduction

Volatile market environments have the characteristics of short product life cycles, uncertain production types, and fluctuating production volumes [1, 2]. It is hard for traditional production systems, such as flow shop, Toyota production system, job shop, and cellular manufacturing system, to adapt to such environments. Seru production system (SPS), which is the latest innovation in production management in the Japanese manufacturing industry, is a new type of production system with quick responsiveness and high flexibility [3]. Since the 1980s, many Asian leading electronic enterprises such as Canon, Samsung, Sony, Panasonic, LG, Fujitsu, and NEC have adopted SPSs to enhance their coping ability to volatile markets [4, 5]. Roth et al. [2] reviewed the last 25 years of the growth and evolution of operations and supply chain management and noted that serus are more flexible than Toyota production systems and represent the next generation of lean. Browning and De Treville [6] also pointed out that seru can be an alternative to the TPS when meeting rapid product proliferation and highly volatile consumer demand.

Many benefits, such as reducing lead time, setup time, working in process inventories, finished product inventories, cost, required workforce, and shop floor space, are gained by adopting SPSs [4, 7]. Many case studies show the significant advantages of SPSs in volatile market environments [3, 8, 9]. For example, in Nagahama Canon, six assembly lines are converted to serus. Profit had increased by 200% even though the sales had decreased from 1,300 hundred million to 1,020 hundred million Yen. The workforce and throughput time had been reduced by 10% and 33%, respectively [3, 4]. Each seru is a small, complete, and human-centered assembly unit. It consists of some cheap equipment and one or more cross-trained workers. An SPS includes at least one seru. Serus in SPSs can be

frequently organized, dismantled, and rebuilt in a short time [1, 3, 4, 10, 11].

The configuration problem of an SPS can be divided into two categories, i.e., seru formation [12] and seru scheduling [13]. Seru formation means grouping workers into serus and determining their tasks according to the arrived orders [12]. Seru scheduling refers to constructing serus with an exact sequence in limited workspace [13].

Many studies focused on seru formation [12, 14–17]. For example, Yu et al. proposed an algorithm based on the nondominated sorting genetic algorithm 2 (NSGA-II) to solve the problem of line-to-seru conversion problem whose objectives are minimizing total throughput time and total labor hours [12]. Yu et al. investigated a seru formation problem toward reducing workers and developed exact and metaheuristic algorithms for the small- to medium-scale instances [16]. Yu et al. considered a line-hybrid seru formation problem by combining the two evaluated performances, i.e., makespan and total labor hours. Exact and heuristic algorithms are proposed to solve the different scale instances [17].

For seru scheduling, there are few studies [13, 18]. Yu et al. selected ten usually used scheduling rules to investigate the performance of different rules on the line-seru conversion problem. Two improved exact algorithms based on reducing time complexity and space complexity are proposed [13]. Sun et al. focused on a seru scheduling problem with minimizing the total tardiness and analysed the solution space. They proposed a cooperative coevolution algorithm for large-scale problems and an exact algorithm for small-scale problems [18].

Most of the above studies have the following assumptions: (1) once serus are constructed, they never change; and (2) workers are fully cross-trained. However, to cope with volatile markets, SPSs should be more flexible and quick responsive than traditional production systems. Just-in-time operation system (JIT-OS) [4], whose goal is configuring correct serus, in the right place, at the appropriate time, and in the exact amount, is proposed as the management and control principle of SPSs. Following this principle, serus should be built up over time rather than fixed. Besides, in real production environments, as an important characteristic and central resource, human workers cannot be neglected and it is hard to achieve that all workers are fully cross-trained. Therefore, if some serus require the same worker, these serus cannot be built simultaneously.

Based on the above consideration, the seru scheduling problem can be seen as a complex variant of the parallel machine scheduling (PMS) problem [5, 19]. Some studies took PMS with resource constraints into consideration [20–22]. Though resource constraints are considered in the above studies, they assumed that the resources are extra resources, which are needed by machines or workers to be able to function. The lack of special resources leads to a lot of waiting time. The above waiting time cannot be reduced by changing the production planning. In our model, the waiting time is caused by human workers needed by different serus and can be reduced by optimizing the scheduling.

Seru scheduling with resource (human workers) conflicts (SSRC) is considered in this study. First, this is an important decision problem in operation management in SPSs, which can reveal the nature of reconfiguration ability and quick response ability of SPSs. Second, from a model perspective, it is a new and realistic variant of the PMS problem.

The contributions of this study are as follows. First, to our knowledge, this is the first work addressing the seru scheduling problem with the consideration of human resource conflicts. Second, genetic programming with structure similarity-based operators (GP-SS), which considers special attributes of the seru scheduling problem, is proposed to automatically generate effective rules to effectively and efficiently solve the SSRC. Third, comparative experiments conducted with human-made rules, basic genetic programming (GP), GP-based algorithm, and state-of-the-art algorithms verify the effectiveness of GP-SS.

The rest of this study is organized as follows. Section 2 describes addressed SSRC problem. The proposed GP-SS is presented in Section 3. Comparative results are shown in Section 4. Section 5 concludes the study in the end.

2. Seru Scheduling Problem with Resource Conflicts

In this study, the SSRC problem is proposed as to how to arrange the sequence of constructing serus with resource conflicts under limited spaces. In particular, a SPS is considered in a manufacturing firm. This firm has some locations to configure serus, and each location can accommodate one seru at a time. When several orders come, serus should be configured to finish these orders. Resource conflicts may exist within some serus, so they cannot be built simultaneously. The sequence of constructing serus should be determined to minimize the makespan of these orders. The proposed problem can be described as follows.

2.1. Assumptions. The SSRC addressed in this study is based on the following assumptions.

(1) The information (arrival times and seru(s) that configured for each order) of orders over a period of time is known when a time period starts. The arrival time of orders may be different.

(2) For every single order, at least one seru should be configured and the information (numbers, life cycles, and conflicts with other serus) of seru(s) for each order is predetermined.

(3) Serus that needs the same worker(s) cannot be built simultaneously.

(4) The build time and dismantle time of serus are neglected.

(5) The number of spaces in a seru production system is limited, and one space accommodates only one seru at a time.

2.2. Notations. The notations adopted to describe the addressed problem are presented below.

2.2.1. Indices. The following indices were used in this study:

$o = 1, 2, O$: index of orders

$i = 1, 2, I$: index of serus

$l = 1, 2, L$: index of the spaces to accommodate serus

$t = 1, 2, T$: time

2.2.2. Parameters. The following parameters were used in this study:

at_o: arrival time of order, $o\mathcal{S}_o$: set of serus that configured for order, and $o\,t_i$: life cycle of seru i from being constructed to dismantled

$$v_i^{i'} = \begin{cases} 1, & \text{serus } i \text{ and } i^A \text{ have the resource conflict,} \\ 0, & \text{otherwise} \end{cases}$$

2.2.3. Decision Variables. The following decision variables were used in this study:

st_i: start time of seru i.

$$x_i^l = \begin{cases} 1, & \text{seru } i \text{ is built in space } l, \\ 0, & \text{otherwise} \end{cases}$$

2.3. Formulation of Seru Scheduling with Resource Conflicts. Seru scheduling with resource conflicts can be formulated using equations (1)–(4).

2.3.1. Objective Function. The objective function

$$\min C_{\max} \tag{1}$$

is subject to

$$\sum_{l}^{L} x_i^l = 1, \forall i, \tag{2}$$

$$\sum_{i}^{I} t_i x_i^l \le C_{\max}, \forall l, \tag{3}$$

$$st_i \ge st_{i'} + t_{i'} or st_{i'} \ge st_i + t_i, if\, v_i^{i'} = 1. \tag{4}$$

Equation (1) states that the objective of the addressed problem is to minimize the makespan of orders in the same time period. $C_{\max} = \max\{st_i + t_i\}, \forall i$, which is the amount of time to process all coming orders. Constraint (2) ensures that each seru is exactly built in one space. Constraint (3) ensures that the makespan is at least as large as the total life cycles of serus built in each space. Constraint (4) means that serus i and i' cannot be built simultaneously.

2.4. Model Complexity. In this section, the NP-hardness of SSRC is proved as follows.

Theorem 1. *SSRC is NP-hard.*

Proof. The PMS problem is a classic NP-hard problem. We reduce any instance of the PMS problem to an instance of SSRC. The PMS problem can be described as follows.

Given a set of jobs, Jobs $= 1, H$, each $h \in$ Jobs is characterized by a release time f_h and a processing time v_h. These jobs are scheduled on M machines, and preemption is not allowed. The target is to minimize the makespan.

Let $st_i = f_h$ ands $t_i = v_h$, where $i = h$, $I = H$, $L = M$. Obviously, a seru is the same as a job in PMS and a location is the same as a machine in PMS. When the solution of the PMS instance is found, the corresponding instance of SSRC can be solved. Thus, the addressed SSRC problem in this study is NP-hard. □

3. Methodology

In this section, GP-SS is proposed to solve the addressed SSRC problem. Studies on seru scheduling employed heuristic rules [13, 18] in which problem instances are small scale. In industrial environments, heuristic rules are often used due to simplicity and efficiency [23, 24]. Though efficient heuristic rules can bring many benefits for manufacturing, they also may lead to malfunctions because of the limitation of human experiences. Besides, the performance of heuristic rules heavily depends on the environments and objectives. For each single heuristic rule, no one can outperform others in all circumstances [23].

Genetic programming (GP) is a kind of technology, which can strongly generate heuristic rules based on specific problem attributes. Because of its high efficiency and generalization ability, GP is used to evolve heuristics in many scheduling domains, such as single machine scheduling [25–27], job shop scheduling [28–32], flow shop scheduling [33], and airplane scheduling [34, 35]. In recent years, GP is also used to solve the PMS problem [36–38]. However, GP has several drawbacks. First, in the evolutionary process, the tree-structured individuals are changed frequently by operations of mutation and crossover, which may lead to large and complex individuals but with poor quality. Thus, GP may waste a lot of computing time in exploiting good solutions. Second, basic operators may not be powerful enough to help GP search for good enough solutions, which means that GP is easy to be trapped into local optima.

Similarity-based crossover operator is a kind of improved special operator in GP, which performs well in real-valued function regression problem [39, 40]. Its design concept in the above studies is that exchange of subtrees is most likely to get benefits if the two subtrees are semantically similar, which means that they are not semantically identical, but also they are not too semantically dissimilar. Semantic similarity refers to the difference value of fitness between two subtrees obtained by sampling methods. It is verified to be a useful strategy to improve the performance of GP and to shorten the sizes of tree-structured individuals. However, in continuous optimization, such as real-valued function regression problem, it requires very little computing time cost to calculate fitness when compared to

scheduling problems. Such a method cannot be directly used to discrete GP to obtain similar benefits. It always uses less time to compare the structure similarity between two subtrees when compared with calculating the fitness. Therefore, in this study, GP with structure similarity-based operators (GP-SS) is proposed to generate effective rules to solve SSRC.

3.1. Framework of GP-SS. The evolutionary process of GP-SS is shown in Figure 1. At first, the initial population consisting of tree-structured individuals is randomly generated. Then, the fitness of individuals is calculated and the good ones are selected by using the roulette wheel method to generate the next generation. The structure similarity-based crossover and mutation operators are adopted to improve the optimization ability and convergence speed of GP-SS. Finally, the termination criterion is used to determine whether to terminate the algorithm.

3.2. Fitness Calculation of GP-SS. In GP-SS, as in basic GP, tree-structured individuals are used. In each individual, its root nodes represent function operators and leaf nodes represent an element in the terminal set. The terminals that are derived from the SSRC problem are shown in Table 1. The function set includes addition (ADD), subtraction (SUB), protected division (DIV), and multiplication (MUL). The function set and terminal set are used to rank serus. The individual SUB (AT, ET) is taken as an example. There are two serus, seru 1 and seru 2, that need to be scheduled. It is assumed that these two serus are both built for one order and that the "AT" attributes of these two serus (arrival time) are set to 100. The "ET" attributes of serus 1 and 2 (life cycle) are set to 50 and 30, respectively. The rule derived from the individual is AT-ET. Thus, the scores of serus 1 and 2 are (100-50) = 50 and (100-30) = 70, respectively. Because the score of seru 1 (50) is less than the score of seru 2 (70), the built sequence of serus 1 and 2 is as follows: seru 1 is prior to seru 2. Then, the fitness of this schedule can be calculated by using equation (1).

3.3. Structure Similarity-Based Operators. In GP-SS, structure similarity-based operators are inspired by similarity-based crossover operator designed for real-valued function regression problem [39, 40]. In order to be suitable for solving discrete optimization problems, a similarity coefficient is proposed in this study, which replaces the similar distance in continuous optimization.

Let $ssc(st_1, st_2)$ be the structure similarity coefficient between subtree 1, st_1, and subtree 2, st_2. $a^u_{st_1}$ and $a^u_{st_2}$ are defined as the number of $u-th$ attribute of N (according to Table 1, $N = 7$ in this study) attributes in subtrees 1 and 2, respectively. Therefore, $ssc(st_1, st_2)$ can be defined by Algorithm 1.

In Algorithm 1, α, β, and γ are parameters in GP-SS, known as the lower bound for attributes' similarity sensitivity, the upper bound for attributes' similarity sensitivity,

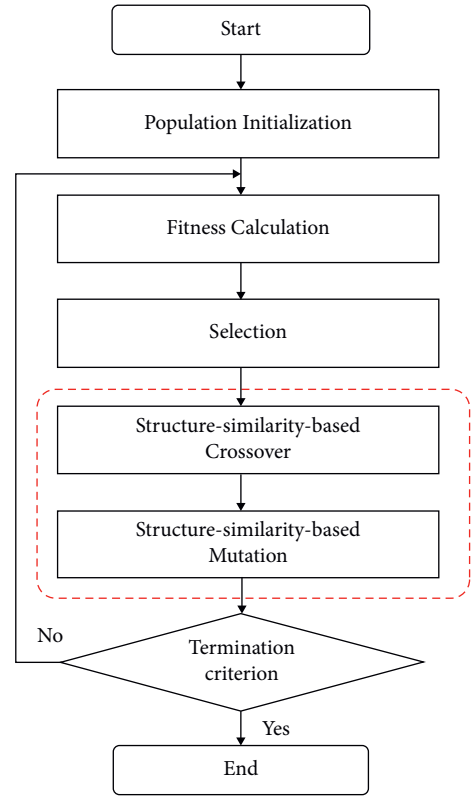

Figure 1: Framework of the GP-SS.

Table 1: Terminals for the SSRC problem.

Symbol	Description
AT	Arrival time of an order that a seru is configured for
ET	Life cycle of a seru
CFL	Number of current free spaces
W	Weight of a seru
OF	A number of serus that are overlapped with a seru
CT	Current time
#	Random number between 0 and 1

and the upper bound for layer difference, respectively. $ssc(st_1, st_2) = 1$ means st_1 is similar to st_2.

The layer difference between the two trees st_1 and st_2 is defined as follows:

$$l\,d(st_1, st_2) = \left| l_{st_1} - l_{st_2} \right|, \tag{5}$$

where $l\,d(st_1, st_2)$ is the layer difference between st_1 and st_2. l_{st_1} and l_{st_2} are the layers of st_1 and st_2, respectively. The layer of a tree can be calculated by traversing the tree.

For example, Figure 2 shows two subtrees st_1 and st_2. st_1 can be transformed into AT + (ET − #/OF), and st_2 can be transformed into W/CFL + AT by in order traversal of these two trees, respectively. The layers of these two subtrees are 4 and 3, so the layer difference between st_1 and st_2 is 1.

Based on the above definition of structure similarity coefficient, the structure similarity-based crossover and mutation can be defined by Algorithms 2 and 3, respectively.

(1) : **if** st_1 and st_2 meet the following conditions:
(2) : condition 1: the layer difference between two trees is less than γ
(3) : condition 2: $\alpha < \sum_n^N |a_{st_1}^u - a_{st_2}^u|/N < \beta$ **then**
(4) : $ssc(st_1, st_2) = 1$
(5) : **else**
(6) : $ssc(st_1, st_2) = -1$
(7) : **end if**

Algorithm 1: Determination of the similarity between subtrees 1 and 2.

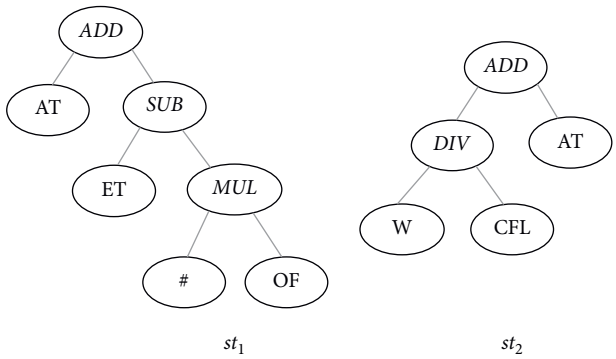

Figure 2: Example of two subtrees. (a) st_1. (b) st_2.

In Algorithm 2, *attemptLimit* is the upper limit of attempts to find similar subtrees and $list_{similarity}$ is a list can be used in structure similarity-based mutation ($list_{similarity}$ is set to be empty at each end of generation). In this process, if the pair of similar subtrees is not found, structure similarity-based crossover is degenerated to classic crossover.

In Algorithm 3, *ran do mPara* is the upper limit parameter that controls the structure similarity-based mutation probability. In the process of structure similarity-based mutation, if *ran do mSee d* is out of predefined range or $list_{similarity}$ is empty, the structure similarity-based mutation is degenerated to classic mutation.

3.4. Computational Complexity Analysis. In this section, the complexity analysis of proposed structure similarity-based operators is given. For any pair of subtrees, suppose that the numbers of their nodes are m and n, the time complexity of Algorithm 1, which traverses all nodes of two subtrees several times, is $O(m + n)$. Thus, the time complexity of proposed structure similarity-based operators is $O(m + n)$.

4. Computational Results and Comparison

To evaluate the performance of the proposed GP-SS algorithm, a series of comparisons are conducted in this section. The GP-SS and other comparison algorithms are shown in Table 2. All experiments are performed on a PC with the i5-10600 processor (4.1 GHz) and 16 GB RAM.

The test problems are generated by setting three important dimensions, which are orders, serus types, and spaces. The number of orders is ranging from 5 to 60, the number of seru types is ranging from 5 to 48, and the number of spaces is ranging from 3 to 35. A notation "3-2-1" is used to represent a test problem with 3 orders, 2 types of serus, and 1 space. For each test problem, 20 independent replications are tested to get the results. Training set and test set are designed to try to overcome overfitting. In the training set, 4 test problems are designed to train the parameters of GP-SS and all comparison algorithms. In the test set, 12 test problems are chosen. All the comparison experiments are run on the test set.

The Gap_x, which is calculated in equation (6), is defined in this section to reflect the performance of GP-SS. This gap is the percentage deviation between the average objective function values obtained by the GP-SS algorithm and the x algorithm for 20 independent replications.

$$\text{Gap}_x = 100 * \frac{\text{score}_x - \text{score}_{\text{GP-SS}}}{\text{score}_{\text{GP-SS}}}, \tag{6}$$

where score_x and $\text{score}_{\text{GP-SS}}$ represent the objective function values obtained by the x algorithm and GP-SS, respectively.

4.1. Parameter Setting. According to the parameter experiments, the parameter values in Table 3 for GP-SS are set as default in the following experiments. Parameters of all comparison algorithms are also trained on the training set.

4.2. Comparison Experiments. To evaluate the performance of GP-SS, 4 groups of experiments are conducted. First, in the previous seru formation and scheduling studies, due to simplicity and efficiency, human-made rules are the most used approach. Therefore, 7 rules are selected to be compared with rules that are generated by GP-SS. Second, to verify the effects of the proposed structure similarity-based operators, GP-SS is compared with basic GP. Third, an excellent variant of GP in the scheduling domain [41], genetic programming hyperheuristic (GP-HH), is chosen to be compared with GP-SS. Finally, two state-of-the-art algorithms designed for PMS, greedy randomized adaptive search procedure with variable neighborhood search (GRASP-VNS) [42] and improved genetic algorithm (IGA) [43], are also chosen to evaluate the performance of our proposed GP-SS.

4.2.1. Comparison with Human-Made Rules. To compare with GP-SS, 5 human-made rules selected from previous studies and 2 special rules designed for SSRC are used. These rules are shown in Table 4.

```
(1) : while count < attemptLimit do
(2) :   Randomly choose the crossover points to generate st_1 and st_2
(3) :   if st_1 is similar to st_2: then
(4) :     crossover and add the children to the new population
(5) :     add st_1 and st_2 into a list list_similarity
(6) :     return 0
(7) :   else
(8) :     count + +
(9) :   end if
(10) : end while
(11) : crossover
```

Algorithm 2: Structure similarity-based crossover.

```
(1) : if ran do mSee d < ran do mPara and list_similarity is not empty then
(2) :   mutate to a randomly chosen subtree in list_similarity
(3) : else
(4) :   mutation
(5) : end if
```

Algorithm 3: Structure similarity-based mutation.

Table 2: Description of GP-SS and comparison algorithms.

Algorithm	Description
GP-SS	This algorithm performs GP with structure similarity-based operators to generate heuristic rules
Human-made rules	Rules are selected from previous studies and are specially designed for SSRC
Basic GP	This algorithm performs GP to generate heuristic rules
GP-HH	This algorithm performs GP with a predesigned low-level heuristics set to generate heuristic rules
GRASP-VNS	Greedy randomized adaptive search procedure with variable neighborhood search
IGA	Genetic algorithm with improved crossover operators and elitism selection mechanism

Table 3: Parameters for GP-SS.

Parameter	Value
α	0.01
β	0.3
γ	3
attemptLimit	5
ran do mPara	0.4
Population size	48
Number of generations	100

Table 4: Seven rules to be compared with GP-SS.

Rule	Description
FCFS	First come first service
LPT	Longest life cycle seru first
SPT	Shortest life cycle seru first
ECT	Earliest completion time seru first
SH	Randomly sequenced serus
HC	Highest conflict factor seru first
LC	Lowest conflict factor seru first

As shown in Table 5, for all test problems, GP-SS outperforms the human-made rules with the range from 5.82% to 133.31%. The reason that GP-SS is superior to human-made rules lies in that human-made rules are predesigned. They can be used in many scheduling domain but cannot be universally good. Rules from GP-SS are specifically evolved for SSRC and can naturally achieve better performance.

4.2.2. Effects of Structure Similarity-Based Operators. To evaluate the performance of structure similarity-based operators, a comparison between GP-SS and basic GP is conducted. The results are shown in Table 6.

As shown in Table 6, for best cases, GP-SS outperforms GP with an average gap of 8.68%. For the average cases, GP-SS outperforms GP with the gap ranging from 20.38% to 36.35%. For the computing time, GP-SS is similar to GP with an average gap of −0.12%. The above results verify that GP-SS is more effective than GP. The reason lies in that some subtrees or structures in individuals may have better effects than others. If these subtrees or structures can be found in the evolutionary process, it can help the algorithm enhance the optimization ability and convergence speed. The design concept of the proposed structure similarity-based operators is giving GP-SS a mechanism to search for a similar structure. Due to the design of two-attempt limit parameters, GP-SS also has the ability to jump from local optimal.

Table 5: Comparison with human-made rules.

Test problem	Gap_{FCFS} (%)	Gap_{LPT} (%)	Gap_{SPT} (%)	Gap_{ECT} (%)	Gap_{SH} (%)	Gap_{HC} (%)	Gap_{LC} (%)
5 – 5 – 3	26.97	36.54	15.53	13.64	14.27	12.39	32.77
10 – 8 – 5	40.8	59.02	16.05	16.78	25.31	5.82	48.75
15 – 12 – 8	28.35	39.10	12.67	13.52	31.55	6.24	46.00
20 – 15 – 10	15.56	47.01	21.94	18.41	25.29	10.35	78.38
25 – 18 – 12	34.76	42.27	13.63	20.54	33.91	8.38	69.79
30 – 21 – 15	24.65	45.11	11.75	19.28	29.35	14.55	85.45
35 – 25 – 18	43.44	24.39	16.88	15.67	17.08	12.09	44.03
40 – 30 – 20	20.58	65.72	21.24	22.91	35.64	14.47	73.05
45 – 30 – 23	43.89	29.33	17.26	17.86	34.11	17.83	84.56
50 – 42 – 27	42.02	46.10	17.64	21.14	34.39	9.27	94.85
55 – 45 – 30	59.61	32.54	16.49	29.20	45.37	13.16	133.31
60 – 48 – 35	41.39	69.19	21.65	26.93	45.02	14.50	76.28
Average gap	35.17	44.69	16.89	19.66	30.94	11.59	72.27

Table 6: Comparison with basic GP.

Test problem	$Gap_{basicGP}$ (%)		
	Min	Avg	Time
5 – 5 – 3	5.52	24.73	−0.03
10 – 8 – 5	4.49	20.38	−2.31
15 – 12 – 8	6.85	30.87	1.53
20 – 15 – 10	7.41	21.45	1.18
25 – 18 – 12	12.36	35.31	−1.58
30 – 21 – 15	7.15	26.69	2.73
35 – 25 – 18	13.20	23.21	−0.64
40 – 30 – 20	7.93	36.35	2.93
45 – 30 – 23	8.49	24.51	−5.31
50 – 42 – 27	13.54	28.89	−3.07
55 – 45 – 30	7.52	26.84	1.52
60 – 48 – 35	9.73	22.73	1.61
Average gap	8.68	26.83	−0.12

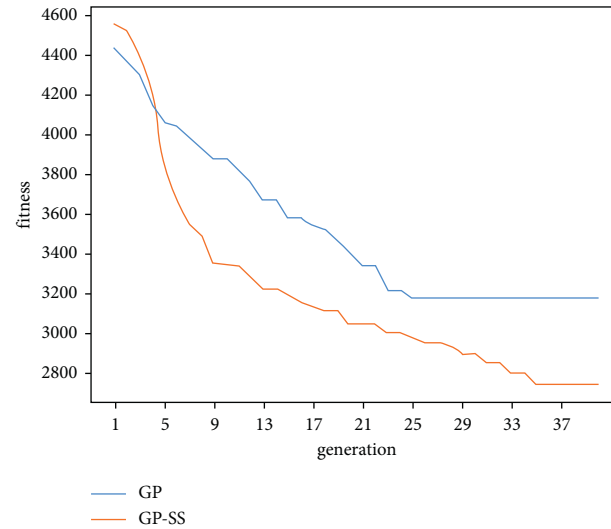

Figure 3: Evolving process of GP-SS and GP.

To further analyze the quick convergence ability of GP-SS, an evolving process from a test problem in the test set is recorded in Figure 3. The vertical and abscissa axes in Figure 3 represent fitness and the index of generation, respectively. It is easy to observe that GP-SS has a faster convergence speed and a better performance. This result can also help to verify the ability to search good regions of GP-SS.

4.2.3. Comparison with GP-Based Algorithm. GP-HH is a hyperheuristic algorithm that employs GP as the high-level strategy to manage several predefined low-level heuristics [41]. The comparison between GP-SS and GP-HH is conducted in this section. The results are shown in Table 7.

As shown in Table 7, for the best cases, the gap between GP-SS and GP-HH ranges from 2.34% to 13.73%, with an average gap of 5.53%. For the average cases, GP-SS outperforms GP-HH with an average gap of 19.99%. For the computing time, GP-SS outperforms GP-HH with an average gap of 4.29%. The results show that GP-SS is more effective and efficient than GP-HH. The reason lies in that low-level heuristics have a high impact on the performance of GP-HH. Without good enough heuristics that are specially designed for SSRC, GP-HH cannot perform much better than basic GP. Therefore, due to the good convergence ability, GP-SS is superior to GP-HH.

4.3. Comparison with Other PMS Scheduling Algorithms. GRASP-VNS [42] and IGA [43] are two recent state-of-the-art algorithms designed for the PMS problem. These two algorithms are compared with GP-SS, and the results are shown in Table 8.

As shown in Table 8, for the best cases, GP-SS performs a little better than GRASP-VNS and IGA with an average gap of 1.35% and 1.93%, respectively. For the average cases, GP-SS outperforms GRASP-VNS and IGA with an average gap of 2.50% and 5.38%, respectively. These results show that, based on the training process on the specific problem and structure similarity-based operators, GP-SS can achieve a slight advantage over other PMS scheduling algorithms.

Meanwhile, the computing time of GP-SS is the sum of the time of generating rules on a training set and applying rules on all test problems. The computing time of GRASP-VNS and IGA is the sum of the time of independently running the algorithm on all test problems. Therefore, the

Table 7: Comparison with GP-HH.

Test problem	Gap_{GP-HH} (%)		
	Min	Avg	Time
5 – 5 – 3	2.95	20.50	0.73
10 – 8 – 5	4.61	30.91	1.61
15 – 12 – 8	2.83	22.94	2.87
20 – 15 – 10	3.25	9.56	1.19
25 – 18 – 12	9.67	19.03	3.85
30 – 21 – 15	7.11	18.65	4.54
35 – 25 – 18	2.34	24.37	2.19
40 – 30 – 20	4.70	7.38	3.78
45 – 30 – 23	5.03	14.63	6.09
50 – 42 – 27	13.73	23.70	5.81
55 – 45 – 30	4.97	16.61	7.92
60 – 48 – 35	5.22	31.64	10.86
Average gap	5.53	19.99	4.29

Table 8: Comparison with GRASP-VNS and IGA.

Test problem	$Gap_{GRASP-VNS}$ (%)		Gap_{IGA} (%)	
	Min	Avg	min	Avg
5 – 5 – 3	0.00	−0.91	0.00	1.30
10 – 8 – 5	0.00	0.05	0.00	6.22
15 – 12 – 8	0.00	0.71	0.00	0.93
20 – 15 – 10	0.00	3.25	1.42	7.31
25 – 18 – 12	0.69	−0.58	1.26	5.22
30 – 21 – 15	1.84	3.93	0.38	7.08
35 – 25 – 18	1.43	2.76	1.95	6.97
40 – 30 – 20	2.61	5.59	2.46	8.42
45 – 30 – 23	1.60	3.68	3.04	6.21
50 – 42 – 27	3.13	7.20	5.68	5.87
55 – 45 – 30	2.17	2.30	3.55	3.07
60 – 48 – 35	2.73	1.99	3.43	5.94
Average gap	1.35	2.50	1.93	5.38

Table 9: The computing time of each m.

Experiment	Gap_{time} (%)
GP-SS vs. GRASP-VNS	64.04
GP-SS vs. IGA	73.21

computing time that is compared in this section is set as the above settings. Because of the big difference in computing time among these three types of algorithms, the gap that shows the percentage deviation between GP-SS and each compared algorithm is given as follows.

$$Gap_{time_x} = 100 * \frac{time_x - time_{GP-SS}}{time_x}, \quad (7)$$

where $time_x$ and $time_{GP-SS}$ represent the computing time obtained by the x algorithm and GP-SS, respectively.

The results are shown in Table 9. It is observed that GP-SS outperforms GRASP-VNS and IGA with the gap of 64.04% and 73.21%, respectively. This result verifies the efficiency of the proposed GP-SS. The reason lies in that GRASP-VNS and IGA need to search the solution space for each test problem. However, GP-SS just needs to search on the training set.

5. Conclusions

In this study, the seru scheduling problem with resource conflicts that minimizes the makespan is considered. In the addressed problem, workers are not necessarily fully cross-trained. Therefore, different serus may need the same worker(s), and serus that need the same worker(s) cannot be built simultaneously. To the best of our knowledge, this is the first work addressing the seru scheduling problem with the consideration of resource conflicts. The NP-hardness of the addressed problem is also proved.

A GP-SS is proposed in this study to automatically generate effective rules to solve the SSRC. In GP-SS, structure similarity-based operators are designed to help GP-SS focus on more promising regions in searching spaces with affordable and small computing time increases. By adopting structure similarity-based operators, GP-SS can achieve fast convergence and avoid generating complex and huge tree-structured rules.

Comparative results with human-made rules, basic GP, GP-based algorithm, and other state-of-the-art algorithms verify the effectiveness of GP-SS. For future work, there are several directions to be considered. Interaction between seru formation and SSRC can be taken into consideration. A cooperative coevolution algorithm can be designed to solve this problem. Besides, in addition to the makespan, some other objectives, such as production costs, can be considered.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported by the National Key Research and Development Program under grant 2018YFB1700602 and the Industrial Internet Innovation and Development Project under grant TC2008031.

References

[1] Y. Yin, K. E. Stecke, and D. Li, "The evolution of production systems from industry 2.0 through industry 4.0," *International Journal of Production Research*, vol. 56, no. 1-2, pp. 848–861, 2018.

[2] A. Roth, J. Singhal, K. Singhal, and C. S. Tang, "Knowledge creation and dissemination in operations and supply chain management," *Production and Operations Management*, vol. 25, no. 9, pp. 1473–1488, 2016.

[3] Y. Yin, K. E. Stecke, M. Swink, and I. Kaku, "Lessons from seru production on manufacturing competitively in a high cost environment," *Journal of Operations Management*, vol. 49-51, no. 1, pp. 67–76, 2017.

[4] K. E. Stecke, Y. Yin, and I. Kaku, "Seru: the organizational extension of JIT for a super-talent factory," *International Journal of Strategic Decision Sciences*, vol. 3, no. 1, pp. 105–118, 2012.

[5] Y. Yu and J. Tang, "Review of seru production," *Frontiers of Engineering Management*, vol. 6, no. 2, pp. 183–192, 2019.

[6] T. R. Browning and S. Treville, "A lean view of lean," *Journal of Operations Management*, vol. 67, no. 5, pp. 640–652, 2021.

[7] M. Abdullah and G. A. Süer, "Consideration of skills in assembly lines and seru production systems," *Asian Journal of Management Science and Applications*, vol. 4, no. 2, pp. 99–123, 2019.

[8] D. Treville, M. Ketokivi, and V. Singhal, "Competitive manufacturing in a high-cost environment: introduction to the special issue," *Journal of Operations Management*, vol. 49-51, no. 1-5, pp. 272–6963, 2017.

[9] C. Liu, Z. Li, and J. Tang, "How seru production system improves manufacturing flexibility and firm performance: an empirical study in China," *Annals of Operations Research*, 2021.

[10] Y. Yin, I. Kaku, and K. Stecke, "The evolution of production systems throughout Canon," *Operations Management Education Review*, vol. 2, pp. 27–40, 2008.

[11] K. E. Stecke, Y. Yin, and I. Kaku, "Production: an extension of just-in-time approach for volatile business environment," *Journal of Obstetrics and Gynaecology Research*, vol. 40, no. 6, pp. 1725–1732, 2014.

[12] Y. Yu, J. Tang, J. Gong, Y. Yin, and I. Kaku, "Mathematical analysis and solutions for multi-objective line-cell conversion problem," *European Journal of Operational Research*, vol. 236, no. 2, pp. 774–786, 2014.

[13] Y. Yu, S. Wang, J. Tang, I. Kaku, and W. Sun, "Complexity of line-seru conversion for different scheduling rules and two improved exact algorithms for the multi-objective optimization," *SpringerPlus*, vol. 5, no. 1, p. 809, 2016.

[14] Y. Yu, J. Tang, W. Sun, Y. Yin, and I. Kaku, "Combining local search into non-dominated sorting for multi-objective line-cell conversion problem," *International Journal of Computer Integrated Manufacturing*, vol. 26, no. 4, pp. 316–326, 2013.

[15] Y. Yu, J. Tang, W. Sun, Y. Yin, and I. Kaku, "Reducing worker(s) by converting assembly line into a pure cell system," *International Journal of Production Economics*, vol. 145, no. 2, pp. 799–806, 2013.

[16] Y. Yu, W. Sun, and J. Tang, "Line-seru conversion towards reducing worker(s) without increasing makespan: models, exact and meta-heuristic solutions," *International Journal of Production Research*, vol. 55, no. 9-10, pp. 2990–3007, 2017.

[17] Y. Yu, W. Sun, J. Tang, and J. Wang, "Line-hybrid seru system conversion: models, complexities, properties, solutions and insights," *Computers & Industrial Engineering*, vol. 103, no. 1, pp. 282–299, 2017.

[18] W. Sun, Y. Yu, Q. Lou, J. Wang, and Y. Guan, "Reducing the total tardiness by Seru production: model, exact and cooperative coevolution solutions," *International Journal of Production Research*, vol. 58, no. 21, pp. 6441–6452, 2019.

[19] C. Charalambous and K. Fleszar, "Variable neighborhood descent for the unrelated parallel machine scheduling problem," *The International Journal on Artificial Intelligence Tools*, vol. 21, no. 04, pp. 1–18, 2012.

[20] L. Fanjul-Peyro, F. Perea, and R. Ruiz, "Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources," *European Journal of Operational Research*, vol. 260, no. 2, pp. 482–493, 2017.

[21] F. Krzysztof and S. Khalil, "Algorithms for the unrelated parallel machine scheduling problem with a resource constraint," *European Journal of Operational Research*, vol. 271, no. 3, pp. 839–848, 2018.

[22] E. Vallada, F. Villa, and L. Fanjul-Peyro, "Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem," *Computers and Operations Research*, vol. 111, no. Nov, pp. 415–424, 2019.

[23] D. Li, R. Zhan, and S. Du, "Automatic design of intercell scheduling heuristics," *IEEE Transactions on Automation Science and Engineering*, vol. 99, pp. 1–15, 2019.

[24] M. Solimanpur, P. Vrat, and R. Shankar, "A heuristic to minimize makespan of cell scheduling problem," *International Journal of Production Economics*, vol. 88, no. 3, pp. 231–241, 2004.

[25] T. P. Adams, *Creation of Simple, Deadline, and Priority Scheduling Algorithms Using Genetic Programming*, Stanford, California, CA, USA, 2002.

[26] C. Dimopoulos and A. M. S. Zalzala, "Investigating the use of genetic programming for a classic one-machine scheduling problem," *Advances in Engineering Software*, vol. 32, no. 6, pp. 489–498, 2001.

[27] C. Dimopoulos and A. M. Zalzala, "A genetic programming heuristic for the one-machine total tardiness problem," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*, vol. 3, pp. 2207–2214, Washington, D.C., USA, July 1999.

[28] D. Jakobovic and L. Budin, "Dynamic scheduling with genetic programming," in *Proceedings of the EuroGP-2006*, pp. 73–84, Budapest, Hungary, April 2006.

[29] K. Miyashita, "Job-shop scheduling with genetic programming," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pp. 505–512, Nevada, NV, USA, July 2000.

[30] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2013.

[31] R. Hunt, M. Johnston, and M. Zhang, "Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 927–934, Vancouver, BC, Canada, July 2014.

[32] K. Morikawa, K. Nagasawa, and K. Takahashi, "Job shop scheduling by branch and bound using genetic programming," *Procedia Manufacturing*, vol. 39, pp. 1112–1118, 2019.

[33] H.-B. Song and J. Lin, "A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times," *Swarm and Evolutionary Computation*, vol. 60, Article ID 100807, 2021.

[34] V. Cheng, L. S. Crawford, and P. K. Menon, "Air traffic control using genetic search techniques," in *Proceedings of the 1999 IEEE International Conference on Control Applications*, vol. 1, pp. 249–254, August 1999.

[35] N. Jain, "Genetic search methods in air traffic control," *Computers and Operations Research*, vol. 31, no. 3, pp. 445–459, 2004.

[36] D. Marko, J. Domagoj, and K. Karlo, "Adaptive scheduling on unrelated machines with genetic programming," *Applied Soft Computing*, vol. 48, pp. 419–430, 2016.

[37] D. Marko and J. Domagoj, "Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment," *Applied Soft Computing*, vol. 96, Article ID 106637, 2020.

[38] J. Kristijan, D. Marko, and J. Domagoj, "Designing dispatching rules with genetic programming for the unrelated machines environment with constraints," *Expert Systems with Applications*, vol. 172, Article ID 114548, 2021.

[39] N. Q. Uy, N. T. Hien, N. X. Hoai, and M. O'Neill, "Improving the generalisation ability of genetic programming with semantic similarity based crossover," in *Proceedings of the 2010 European Conference on Genetic Programming-EuroGP 2010*, pp. 184–195, Istanbul, Turkey, April 2010.

[40] Q. U. Nguyen, X. H. Nguyen, and M. O'Neill, "Semantic aware crossover for genetic programming: the case for real-valued function regression," in *Proceedings of the 2009 European Conference on Genetic Programming-EuroGP2009*, Tübingen, Germany, April 2009.

[41] J. Lin, L. Zhu, and K. Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Expert Systems with Applications*, vol. 140, Article ID 112915, 2020.

[42] S. Báez, F. Angel-Bello, A. Alvarez, and B. Melián-Batista, "A hybrid metaheuristic algorithm for a parallel machine scheduling problem with dependent setup times," *Computers & Industrial Engineering*, vol. 131, no. 5, pp. 295–305, 2019.

[43] Q. V. Dang, T. V. Diessen, and T. Martagan, "A matheuristic for parallel machine scheduling with tool replacements," *European Journal of Operational Research*, vol. 291, no. 2, pp. 640–660, 2020.