Int J Adv Manuf Technol (2010) 50:729–747
DOI 10.1007/s00170-010-2518-5

ORIGINAL ARTICLE

Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems

Li Nie · Xinyu Shao · Liang Gao · Weidong Li

Received: 9 July 2009 / Accepted: 4 January 2010 / Published online: 4 February 2010

Abstract The paper considers the problems of scheduling *n* jobs that are released over time on a machine in order to optimize one or more objectives. The problems are dynamic single-machine scheduling problems (DSMSPs) with job release dates and needed to be solved urgently because they exist widely in practical production environment. Gene expression programming-based scheduling rules constructor (GEPSRC) was proposed to construct effective scheduling rules (SRs) for DSMSPs with job release dates automatically. In GEPSRC, Gene Expression Programming (GEP) worked as a heuristic search to search the space of SRs. Many experiments were conducted, and comparisons were made between GEPSRC and some previous methods. The results showed that GEPSRC achieved significant improvement.

Keywords Single machine scheduling · Dynamic scheduling · Release dates · Scheduling rules · Gene expression programming

1 Introduction

Scheduling plays an important role in a shop floor control system, which has a significant impact on the performance of the shop floor. Scheduling is to allocate scarce resources (usually are machines) to activities (usually are jobs) with the objective of optimizing one or more performance criteria (for instance, minimizing makespan, flow time, lateness, or tardiness) [1]. In recent years, more and more effective scheduling methods for shop floor control have emerged with the developments in scheduling methodologies (in research and in practice) as well as technological advances in computing.

Scheduling problems investigated by researchers for several decades may be categorized roughly into two types, static scheduling problems and dynamic scheduling problems. In static scheduling problems, it is usually assumed that the attributes of all jobs to be scheduled are available simultaneously at the start of the planning horizon and unchanged during the planning horizon. The assumption is made mainly for the sake of convenience to model the system considered and solve the scheduling problems that exist. However, the assumption does not always accord with the practical production environment, since there are always all kinds of random and unpredictable events that occur. For example, jobs arrive continuously over time, machines break down and are repaired, and the due dates of jobs are changed during processing. It is rarely possible that the attributes of all jobs to be scheduled are available at the start of planning horizon and unchanged during the horizon. Most scheduling problems that exist in such environment are called dynamic scheduling problems [2]. Dynamic scheduling problems have attracted more and more attention. For example, Kianfar et al. [3] studied a flexible flow shop system considering dynamic arrival of jobs; Wang et al. [4] considered the single-machine scheduling problem with a deteriorating function, which means that the actual job processing time is a function of jobs already processed; Ham and Fowler [5] considered the scheduling of batching

L. Nie · X. Shao · L. Gao (✉)
The State Key Laboratory of Digital Manufacturing Equipment and Technology,
Huazhong University of Science and Technology,
Wuhan, Hubei 430074, People's Republic of China
e-mail: gaoliang@mail.hust.edu.cn

W. Li
Faculty of Engineering and Computing, Coventry University,
Coventry, UK

operations with job release dates in wafer fabrication facilities. Although some of static scheduling problems are often solvable exactly in polynomial time, most of them are NP-hard. Dynamic scheduling problems are usually more difficult to solve than static ones.

The paper considers the problems of scheduling n jobs that are released over time on a machine in order to optimize one or more objectives, which are dynamic single-machine scheduling problems (DSMSPs) with job release dates. The problems are considered for the following reasons: First, dynamic scheduling problems exist widely in practice and need to be solved urgently, although they pose bigger challenges than static scheduling problems. Secondly, single-machine scheduling problems often form components of solutions for more complex scheduling environments. For example, a job shop scheduling problem may be decomposed into single-machine sub-problems [6].

Static scheduling problems have been studied for almost half of a century, and many effective methods have been proposed. At the beginning, many enumerative-based techniques have been developed. Although enumerative methods such as branch and bound usually provide optimal solutions, the cost of computation time is huge even for a moderate size problem [7]. In the last decades, therefore, many heuristic methods, including dispatching rules [8] and search-based methods, such as simulated annealing [9], tabu search [10], and genetic algorithms (GAs) [11] have been developed to solve larger problems in a reasonable time. Search-based methods usually offer high-quality solutions. However, neither enumerative-based techniques nor search-based methods are appropriate in dynamic conditions, because once the schedule is prepared, the processing sequence of all jobs is determined, and it is inevitable to modify the schedule frequently to respond to the change of the system.

Over the last two decades, much effort has been made to propose new strategies or approaches to solve dynamic scheduling problems. Aytug et al. [12] categorized roughly existing strategies into three classes: completely reactive approaches, robust scheduling approaches, and predictive–reactive approaches. Completely reactive approaches have been widely employed in a large number of scheduling systems and formed the backbone of much industrial scheduling practice. The approaches are characterized by least commitment strategies such as real-time dispatching that create partial schedules according to the current state of the shop floor and the production objectives. Many heuristics, also called dispatching rules, are frequently used to examine the jobs waiting in processing at the given machine or those that arrive in the immediate future, at each time t when the machine is idle and to compute a priority value for each job. The next job to be processed is selected from them by sorting and filtering them according to the priority values assigned to them and selecting the job at the head of the resulting list. The priority function which is encapsulated in the heuristic and assigns values to jobs is usually called with the term scheduling rules (SRs) [1].

Several important achievements on DSMSPs with job release dates are reviewed below. An online algorithm to minimize makespan problem, now commonly called list scheduling, was firstly investigated by Graham [13]. It is a simple greedy algorithm and does not use the information about processing times of jobs. Similar to the work of Graham, other researchers made other research on the online heuristics and achieved many results. As for the total completion time problem, if all jobs are released at the same time, Smith showed that the problem can be solved optimally by the well-known shortest processing time (SPT) rule [14]. For the preemptive version, Baker's work showed that it is easy to construct an optimal schedule online by always running the job with shortest remaining processing time (SRPT) [15]. In the case of single-machine non-preemptive scheduling for minimizing the total completion time, Hoogeveen and Vestjens [16] gave online 2-approximation algorithms, called delayed SPT rule, and proved that the lower bound on online scheduling is 2. Phillips et al. [17] gave a different 2-competitive algorithm called PSW algorithm, which converts preemptive schedules to non-preemptive schedules while only increasing the total completion time by a small constant factor. It is noticeable that it was not the average flow time of a set of jobs that was studied in the literature. Although average flow time is equivalent to average completion time at optimality, Kellerer et al. [18] have shown that the approximability of these two criteria can be quite different. Guo et al. [19] modified the PSW algorithm to solve minimizing total flow time on a single machine with job release dates and proved that this new algorithm yields good solutions for the problem on average. Other objective functions were rarely considered under the model of the dynamic single-machine scheduling with job release dates. For a review on online scheduling results, the comprehensive reviews of [20, 21] are referred. Apart from these simple online heuristics, other classical scheduling rules were also reported in literatures, which are the results of decades of research [22].

The general conclusion on scheduling rules is that no rule performs consistently better than all other rules under a variety of shop configurations, operating conditions, and performance objectives, because the rules have all been developed to address a specific class of system configurations relative to a particular set of performance criterion and generally do not perform well in another environment or for other criteria. Therefore, many researchers made effort to exploit several methods based on artificial intelligence to learn to select rules dynamically according to the change of the system's state from a number of candidates. For example, Jeong and Kim [23] and Yin and Rau [24] used simulation approach; Chen and Yih [25] and El-Bouri and Shah [26] used neural network; Aytug et al. [27] used genetic learning approach; Trappey et al. [28] used expert systems; Singh et al. [29] used the approach of identifying the worst performing criterion; and Yang and Yan [30] used adaptive approach. These methods are mainly based on learning to select a given rule from among a number of candidates rather than identifying new and potentially more effective rules. However, significant breakthrough beyond current applications of artificial intelligence to production scheduling have been made by other researchers who made it possible to automatically construct effective rules for a given scheduling environments. One of the typical works was the learning system SCRUPLES proposed by Geiger et al. [31]. The system combined an evolutionary learning mechanism, i.e., Genetic Programming (GP) [32], with a simulation model of the industrial facility under study, which automates the tedious process of developing new scheduling rules for a given environment which involves implementing different rules in a simulation model of the facility under study and evaluates the rules through extensive simulation experiments. Other existing similar researches include: Dimopoulos and Zalzala [33] evolved rules with GP for single-machine tardiness problem; Yin et al. [34] evolved rules with GP for single-machine scheduling subject to breakdowns; Jakobovic and Budin [1] evolved rules with GP for dynamic single machine and job shop problem; Atlan et al.[35] and Miyashita [36] applied GP mainly to classic job shop tardiness scheduling; and Tay and Ho [37-39] focused on evolving rules with GP for flexible job shop problem. The characteristic shared by these works is that it is the space of algorithms but not that of potential scheduling solutions is searched with an evolutionary learning mechanism. The point in the space of potential scheduling solutions presents only a solution to the specific scheduling instance, which means that a new solution must be found for different initial conditions. While the point in the space of algorithms represents a solution for all of the problems, instances in a scheduling environment with an algorithm can be used to generate a schedule [1]. However, these GP-based approaches mentioned above have a huge cost on computation time, and the constructed rules are usually formulized complexly.

In this research, gene expression programming-based SR constructor (GEPSRC) was proposed to automatically discover effective SRs for DSMSPs with job release dates. Gene Expression Programming (GEP), one of the evolutionary algorithms, worked as a heuristic search to search the space of algorithms but not that of potential scheduling solutions. The proposed approach was evaluated in a variety of single-machine environment where the jobs arrive over time. GEP was usually possible to discover rules that are competitive with those evolved by GP and the classical heuristics selected from literature. In addition, the computation requirement for training GEP to discover high performing rules is much less than that of GP.

The remainder of the paper is organized as follows. Section 2 gives the statement of the DSMSPs with job release dates. Section 3 describes the heuristic for the scheduling problems. Section 4 proposes the framework and mechanism of the GEPSRC and describes the application of GEPSRC on the scheduling problems. An extensive computational study is conducted within the single-machine environment to assess the efficiency and robustness of the autonomous SRs constructing approach. The experiments and results are provided in Section 5. Section 6 is the conclusion and future work.

2 Statement of dynamic single-machine scheduling problems

The DSMSPs with job release dates is described as follows. The shop floor consists of one machine and n jobs, which are released over time and are processed once on the machine without preemption. Each job can be identified with several attributes, such as processing time p_i, release date r_i, due date d_i, and weight w_i, which denotes the relative importance of job i, i=1,…, n. The attributes of a job are unknown in advance unless the job is currently available at the machine or arrive in the immediate future. It is also assumed that the machine cannot process more than one job simultaneously. The scheduling objective is to determine a sequence of jobs on the machine in order to minimize one or more optimization criteria, in our case, makespan, total flow time, maximum lateness, and total tardiness, respectively. For the sake of convenience, we assume all jobs relatively equal, i.e., w_j=1. Then, the four performance criteria considered are defined below.

$$C_{\max} = \max(c_i, i = 1, ..., n) \tag{1}$$

$$F = \sum_{i=1}^{n} (c_i - r_i) \tag{2}$$

$$L_{\max} = \max(c_i - d_i, i = 1, ..., n) \tag{3}$$

$$T = \sum_{i=1}^{n} \max(c_i - d_i, 0) \tag{4}$$

where c_i denotes the finishing time of job i. $C_{\max}$, F, $L_{\max}$, and T denote makespan, total flow time, maximum lateness, and total tardiness of a problem instance, respectively.

Since GEP is used to search the space of algorithms but not that of potential scheduling solutions in the paper; the scheduling algorithms are evaluated on a large number of training sets or test sets of problem instances, which represent different operating conditions relative to different performance criteria, respectively. In order for all the training sets or test sets to have a similar influence to the overall performance estimation of an algorithm, average criterion value over the training set or test set of problem instances are defined as below.

$$|C_{\max}| = \frac{1}{t}\sum_{j=1}^{t}\frac{C_{\max,j}}{n_j \cdot \overline{p}_j} = \frac{1}{t}\sum_{j=1}^{t}\frac{\max\left(c_{ij}, i = 1, \ldots, n_j\right)}{n_j \cdot \overline{p}_j} \tag{5}$$

$$|F| = \frac{1}{t}\sum_{j=1}^{t}\frac{F_j}{n_j \cdot \overline{p}_j} = \frac{1}{t}\sum_{j=1}^{t}\frac{\sum_{i=1}^{n_j}\left(c_{ij} - r_{ij}\right)}{n_j \cdot \overline{p}_j} \tag{6}$$

$$|L_{\max}| = \frac{1}{t}\sum_{j=1}^{t}\frac{L_{\max,j}}{n_j \cdot \overline{p}_j} = \frac{1}{t}\sum_{j=1}^{t}\frac{\max\left(c_{ij} - d_{ij}, i=1,\ldots,n_j\right)}{n_j \cdot \overline{p}_j} \tag{7}$$

$$|T| = \frac{1}{t}\sum_{j=1}^{t}\frac{T_j}{n_j \cdot \overline{p}_j} = \frac{1}{t}\sum_{j=1}^{t}\frac{\sum_{i=1}^{n_j}\max\left(c_{ij} - d_{ij}, 0\right)}{n_j \cdot \overline{p}_j} \tag{8}$$

where $C_{\max,j}$, F_j, $L_{\max,j}$, and T_j denote the makespan, total flow time, maximum lateness, and total tardiness of problem instance j, respectively; n_j denotes the number of job in problem instance j; $\overline{p}_j$ denotes the mean processing time of all jobs in problem instance j; c_{ij}, r_{ij}, and d_{ij} denote completion time, release date, and due date of job i in problem instance j, respectively; t denotes the number of instances in a training set or test set; and $|C_{\max}|$, $|F|$, $|L_{\max}|$, and $|T|$ represent the average value of makespan, flow time, maximum lateness, and tardiness over the training set or test set of problem instances. It is obvious that algorithms with less objective values of $|C_{\max}|$, $|F|$, $|L_{\max}|$, and $|T|$ are better.

3 Heuristic for DSMSPs with job release dates

In static circumstance, since the attributes of all jobs to be scheduled are available at the beginning of planning horizon (referred to be time 0) and unchanged during the planning horizon, the whole schedules usually can be made at the beginning. However, it is not convenient in dynamic conditions where jobs arrive over time and the release dates cannot be known in advance. At any time, some jobs have arrived and others may arrive in some future moment. In this section, we describe a heuristic for the scheduling problems on a single machine with job release dates, and the release dates are unknown in advance unless the jobs will arrive in the immediate future. This heuristic was proposed firstly by Jakobovic and Budin [1].

Heuristic for DSMSPs with job release dates:

Initialize $t = 0$, where the machine is idle at time t;

While there are unscheduled jobs **do**

  $JS_s(t)$ = {all jobs satisfied with $wt_j < P_{min}(t)$ };

  Calculate priority values for all the jobs in $JS_s(t)$ according to a **SR**;

  Schedule the job with the best priority on the machine, and denote the job with J^*;

  Update the machine's idle time, i.e. t = the completion time of J^*;

End while.

Where $\mathrm{JS}_s(t)$ represents the set of the jobs to be taken into consideration for scheduling at time t; wt_j denotes the waiting time for the arrival of the job j, i.e., $\mathrm{wt}_j = \max\{r_j - t, 0\}$; $P_{\min}(t)$denotes the shortest processing time of the jobs that already arrived but are unscheduled at time t.

It is obvious that the $\mathrm{JS}_s(t)$ consists of two types of jobs: the jobs that already arrived but are unscheduled at time t (denoted with *AType*) and those that are expected to arrived soon and satisfy $\mathrm{wt}_j < P_{\min}(t)$ at time t (denoted with *BType*). It is the SR encapsulated in the heuristic that is responsible for evaluating the priority value for each *AType* job or *BType* job.

It is noticeable that the "best priority" may be defined as the one with the greatest or the lowest value. In the paper, we define that the job with a lower priority value has better priority.

The heuristic for DSMSPs with job release dates is employed in the paper for the reasons as below. First, both the arrived jobs (*AType* job) and the jobs that are expected to arrive soon (*BType* jobs) are taken into consideration for scheduling, which contributes to make a more reasonable scheduling decision. In practical scheduling environments, a job can be identified if it arrives in the immediate future. To take jobs that are expected to arrive in the immediate future (*BType* jobs) into consideration provides a more global perspective for scheduling manager. Second, it dramatically decreases the computation cost for estimating priority values to exclude the jobs with $\mathrm{wt}_j \geq P_{\min}(t)$ from the set of $\mathrm{JS}_s(t)$. For any regular scheduling criterion, such as minimizing makespan, flow time, maximum lateness, and tardiness, the jobs with $\mathrm{wt}_j \geq P_{\min}(t)$ should not be scheduled next. As for the jobs with $\mathrm{wt}_j \geq P_{\min}(t)$, the earliest possible starting time of processing are not earlier than $P_{\min}(t)+t$, which is the earliest possible completion time of the jobs that are currently available. If one of the jobs with $\mathrm{wt}_j \geq P_{\min}(t)$ is selected as the next job to be loaded on the machine at time t, the arrived job whose processing time is $P_{\min}(t)$ could be loaded before the selected job without deteriorating the performance criterion value. In other words, it makes no improvement on the performance criterion value and increases the computation time consumed to take the jobs with $\mathrm{wt}_j \geq P_{\min}(t)$ into consideration for scheduling.

In the heuristic, the SR is the important component, and its behavior makes a significant effect on the performance of the scheduling [1]. In the following section, we describe the method of using GEP to automatically construct SRs which would yield good results considering given heuristic for DSMSPs with job release dates and given performance criterion.

4 GEPSRC

We propose here GEPSRC which discovers effective SRs for DSMSPs with job release dates automatically. As one of the evolutionary algorithms, GEP works as a heuristic search technique to search the space of algorithms or space of SRs for a given scheduling environment but not that of potential scheduling solutions for a specific problem instance. In this section, the framework of GEPSRC is proposed first, and then the application of GEPSRC on the scheduling problems is described in detail.

4.1 Framework of GEPSRC

GEPSRC integrates a learning module with a simulation module of the industrial facility under study in order to automate the process of implementing different rules and evaluating their performance using the simulation experiments. The simulation module works as a performance evaluator, and the learning module uses GEP as its reasoning mechanism to evolve SRs based on the evaluating results passed back from the simulation module. The framework of GEPSRC is shown in Fig. 1.

GEPSRC starts with an initial population which consists of a number of candidate scheduling rules that are randomly generated. These rules are passed to the simulation module that describes the production environment and assessed using one or more quantitative measures of performance. Then, the values of the performance measures for all candidate rules are passed back to the learning module, where the next population of rules is reproduced and modified from the current high-performing rules using evolutionary search operators such as selection with elitism strategy, replication, mutation, and transposition (see Section 4.2.3). This next set of rules is then passed to the simulation module so that the performance of the new rules can be evaluated. This cycle is repeated until the termination condition is satisfied.

4.2 Application of GEPSRC on DSMSPs with job release dates

The reasoning mechanism to explore the space of possible SRs in GEPSRC is GEP. GEP is a new technique of creating computer programs based on principle of evolution, firstly proposed by Ferreira [40]. Like GAs [11] and GP [32], it is also an evolutionary algorithm as it uses populations of individuals, selects them according to fitness, and introduces genetic variation using one or more genetic operators [40]. GEP is a genome/phenome evolutionary algorithm, which combines the simplicity of GAs and the abilities of GP [40]. In a sense, GEP is a generalization of GAs and GP [41]. GEP uses fixed length, linear strings of chromosomes (genome) to represent expression trees (ETs) of different shapes and sizes (phenome), which makes GEP more versatile than other evolutionary algorithms [40]. Each chromosome is composed of elements from functions set (FS) and terminal set (TS) relevant to a particular problem domain. The set of available elements is defined a priori. All of the chromosomes that can be constructed using the element set compose the search space.

The remainder of the section presents the design of FS and TS, mapping mechanism between GEP chromosomes and SRs, and operation of evolutionary search operators of GEP and fitness function.

4.2.1 Designing of FS and TS

Each chromosome of GEP is generated randomly at the beginning of the search and modified during evolutionary

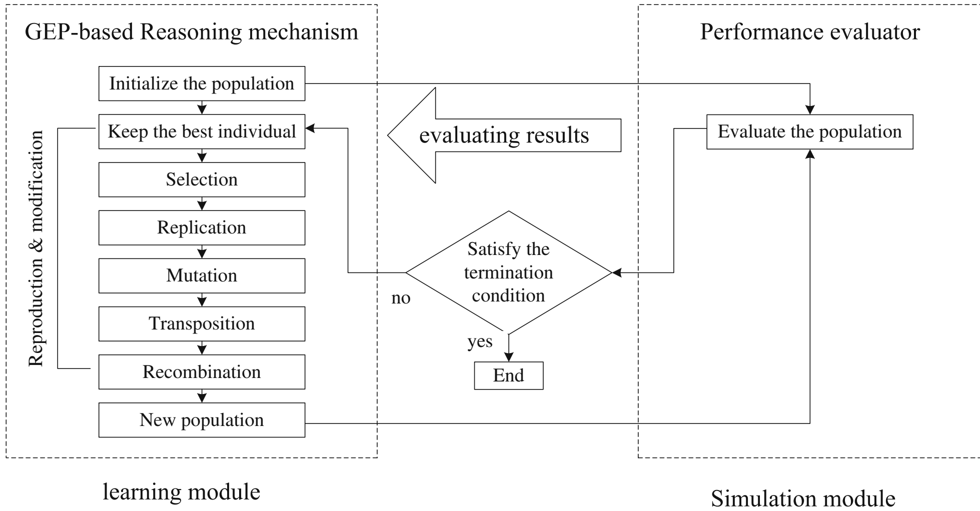


Fig. 1 Framework of GEPSRC

progress with the elements from FS and TS. In other words, GEP uses this predefined set of elements to discover possible solutions for the problem at hand. Therefore, the choice of proper elements for FS and TS is a crucial step in the implementation of optimization process.

The FS and TS used to construct SRs in GEPSRC are defined as follows:

Function Set: including functions such as "+," "–," "*," which express the corresponding arithmetic functions, respectively, and "/" which expresses the protected division which returns 1 when the denominator is equal to 0.

Terminal Set: including elements that denote the current status and attributes of the candidate jobs for scheduling, such as:

- p job's processing time;
- r job's release date;
- d job's due date;
- sl job's positive slack, max $\{d - p - \max\{t, r\}, 0\}$, where t denotes the idle time of the machine;
- st job's stay time, max $\{t - r, 0\}$, where t is defined as above;
- wt job's wait time, max $\{r - t, 0\}$, where t is defined as above.

As for any job, st and wt cannot be positive number simultaneously; st is always 0 for *BType* jobs, while wt is always 0 for *AType* jobs. It is obvious that selecting these two elements to construct SRs is beneficial to make a difference between *AType* jobs and *BType* jobs.

Many researchers incorporate wait time wt into processing time p of a job, i.e., the original processing time of a job is changed to p + wt, which make many SRs designed for static scheduling environment valid to evaluate *BType* jobs in dynamic scheduling environment [1]. However, the method is based on the hypothesis that the wait time information only has an effect on the job's processing time. But it is not always true. Maybe it includes implicitly some information that contributes to make a better decision for the optimization of scheduling process. Therefore, wait time wt is used as one of the potential elements to construct SRs in our work to test GEPSRC's ability to learn and discover new and interesting relationships relative to waiting time which may not be obvious.

It is noticeable that the several elements such as d and sl are important for performance measure of lateness and tardiness but irrelevant to the performance criterion of makespan and flow time. They should be excluded from TS when GEPSRC run for scheduling objective of makespan and flow time. However, as for a scheduling problem in a specific environment relative to a certain criterion, it is usually unknown in advance which attributes of the system might be relative to the objective of scheduling. The aim of including irrelevant elements into the TS in the paper is to examine the ability of GEPSRC to exclude the irrelevant elements in the construction of SRs.

4.2.2 Mapping mechanism between GEP chromosomes and SRs

A SR is actually a mathematic formula which can be encoding into a chromosome of GEP, which typically comprises one or more genes.

A gene in GEP is a fixed length symbolic string with a head and a tail. Each symbol is selected from FS or TS. The symbols which come from FS mean perform a certain operation on arguments. For example, "+" adds two arguments and returns the sum of them. The symbols come from TS have no arguments. For example, "*a*" directly returns the value of the variable *a*. It is stipulated that the head of gene may contain symbols from both the FS and the TS, whereas the tail consists only of symbols come from TS. Suppose the symbolic string has h symbols in the head, and t symbols in the tail, then the length of the tail is determined by the equation $t=h*(n-1)+1$, where n is the maximum number of arguments for all operations in FS, which ensure the correctness of gene, in other words, ensure the validity of the computer program's output [41]. Suppose we use $h=6$ and $n=2$ for arithmetic operations. Thus, the tail length must be $t=7$. So the total gene length is 13.

Consider the FS={+, −, *, /} and the TS={p, r, d, sl, st, wt} (defined in section 4.2.1); a randomly generated GEP gene with size 13 is shown in Fig. 2a. The tail is underlined. The gene can be mapped into an ET shown in Fig. 2b following a depth-first fashion [41]. Specifically, first element in gene corresponds to the root of the ET. Then, below each function is attached as many branches as there are arguments to that function. A branch of the ET stops growing when the last node in this branch is a terminal. The ET shown in Fig. 2b can be further interpreted in a mathematical form as Fig. 2c. It is noticeable that there exist a number of redundant symbols in genes, which are not useful for the gene-ET mapping (genome–phenome mapping). In the example gene, only the first nine symbols are used to construct the ET. The first nine symbols form its valid *K-expression*. The rest are called its *non-coding region*. It is the non-coding region that makes the GEP paramount different from GAs and GP, which always guarantee to product valid new chromosomes, even if any genetic operators are applied on it without restrictions [40].

A typical GEP chromosome which comprises three genes with size of 13 (shown in Fig. 3a) is shown in Fig. 3b, where "|" is used to separate individual genes and underlines are used to indicate the tails. Each gene codes for a sub-ET and the sub-ETs interact with each other in a way of addition to form a more complex multi-subunit ET shown in Fig. 3c. The multi-subunit ET can be explained in a mathematical form as shown in Fig. 3d. It is noticeable that the lengths of *K-expression* of the three genes are 9, 9, and 5, respectively, and the lengths of *non-coding region* of the three genes are 4, 4, and 8, respectively.

4.2.3 Evolutionary search operators

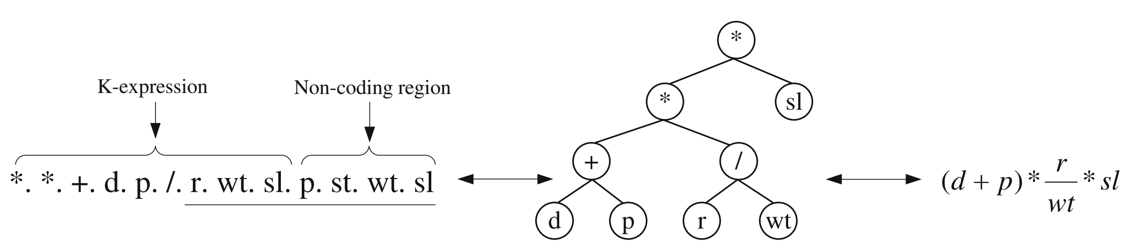
A variety of evolutionary search operators were designed to introduce genetic diversity in GEP population [40].

Selection with elitism strategy Individuals are selected according to fitness by roulette wheel sampling coupled with the cloning of the best individual (simple elitism).

Replication The chromosome is unchanged and enters the next generation directly. The selected individuals are copied as many times as the outcome of the roulette wheel. The roulette is spun as many times as there are individuals in the population in order to maintain the population size unchanged.

Mutation Randomly change symbols in a chromosome. In order to maintain the structural organization of chromosomes, in the head, any symbol can change into any other function or terminals, while symbols in the tail can only change into terminals.

Transposition Randomly choose a fragment of a chromosome and insert it in the head of a gene. The fragment usually consists of several successive symbols in a chromosome. In order not to affect the tail of the gene, symbols are removed from the end of the head to make room for the inserted string. In GEP, there are three kinds of transposition: (1) IS transposition, i.e., randomly choose a fragment begins with a function or terminal (called IS elements) and transpose it to the head of genes, except for the root of genes; (2) RIS transposition, i.e., randomly choose a fragment begins with a function (called RIS elements) and transpose it to the root of genes; (3) gene transposition, i.e., one entire gene in a chromosome is randomly chosen to be the first gene. All other genes in the chromosome are shifted downwards to make place for the chosen gene. Consider the three-genic chromosome in Fig. 4 (the tail is underlined): (a) suppose the fragment "p./.−." in gene 2 is chosen to be an IS element and inserted in the bond 2 in gene 1, then a cut is made in bond 2 and the fragment "p./.−." is copied into the site; the last three



(a) gene/genome (b) ET/phenome (c) mathematical formula

Fig. 2 Mapping between gene and ET in GEP

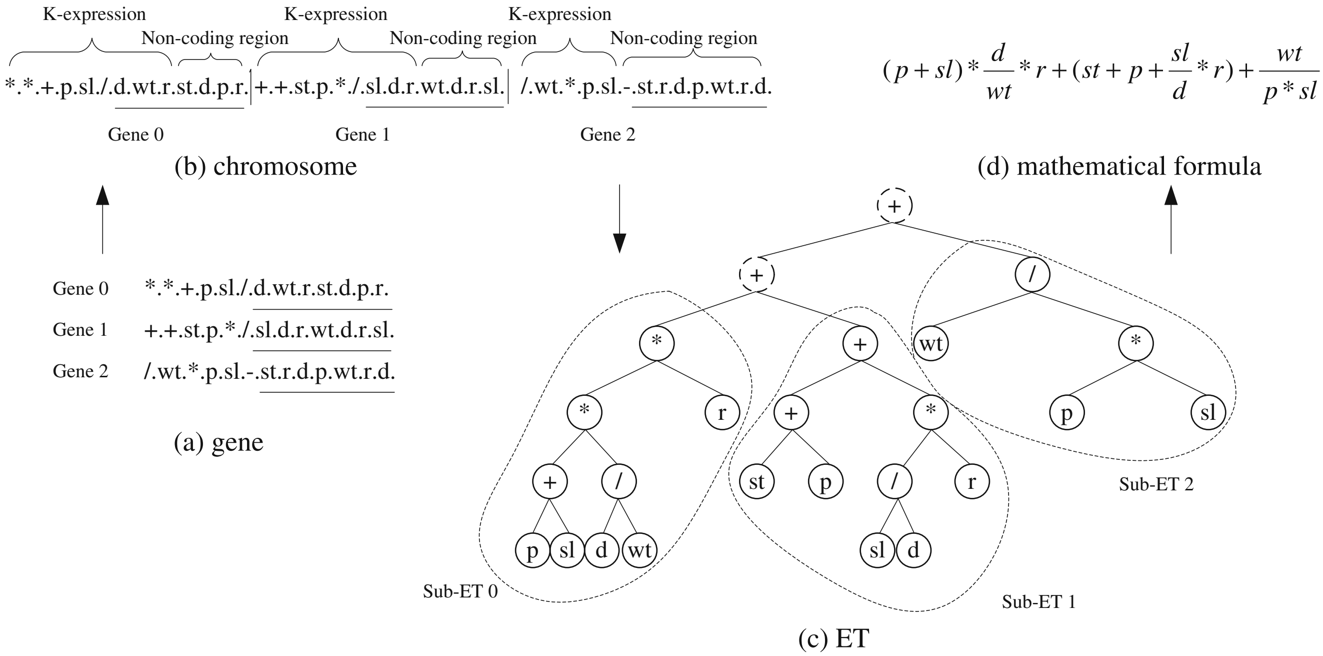


Fig. 3 Mapping between chromosome and SR

symbols in the head are deleted. (b) Suppose the fragment "/.+.d." in gene 0 is chosen to be an RIS element. Then, a copy of the fragment "/.+.d." is made into the root of the gene. The last three symbols in the head are removed. (c) Suppose gene 2 was chosen to undergo gene transposition and moved to the beginning of the chromosome.

Recombination Exchange some material between two randomly chosen parent chromosomes. There are three kinds of recombination: (1) one-point recombination, i.e., split the chromosomes into halves and swap the corresponding sections; (2) two-point recombination, i.e., split the chromosomes into three portions and swap the middle one; (3) gene recombination, i.e., choose one entire gene and swap it between chromosomes.

These genetic operators not only always produce syntactically correct offspring but also are good at creating genetic variation. Mutation and transposition have a tremendous transforming power and usually drastically reshape the ETs. Recombination is excellent for preserving the promising fragment of the sequence of a chromosome to offspring without any constraints.

4.2.4 Fitness function

The fitness function used to evaluate the chromosome of GEP is defined below:

$$f_i = \frac{O_{\max} - O_i}{O_{\max} - O_{\min}} \tag{9}$$

f_i denotes the fitness of the chromosome i, O_i represents the average criterion value over the training set or test set of problem instances obtained with the SR correspondent to chromosome i. $O_{\max}$ and $O_{\min}$ denote the maximal and minimal average criterion value over the same training set or test set of problem instances obtained with the chromosomes of the population, respectively. Since the scheduling objection is minimization, the better chromosome is assigned the bigger fitness.

5 Experiments and results

5.1 Control parameter settings

The reasonable settings for the parameters in GEPSRC are determined through extensive experiments, including: population size, termination condition, number of gene in a chromosome, the length of the head of a gene, mutation rate, IS transposition rate, RIS transposition rate, gene transposition rate, one-point recombination rate, two-point recombination rate, and gene recombination rate. In addition, in the transposition, three transpositions with lengths 1, 2, and 3 were used. Based on these results, the control parameter settings shown in Table 1, column 1 is used in GEPSRC.

5.2 Benchmark heuristics

In order to evaluate the effectiveness of GEPSRC, eight frequently used classical online heuristics and scheduling

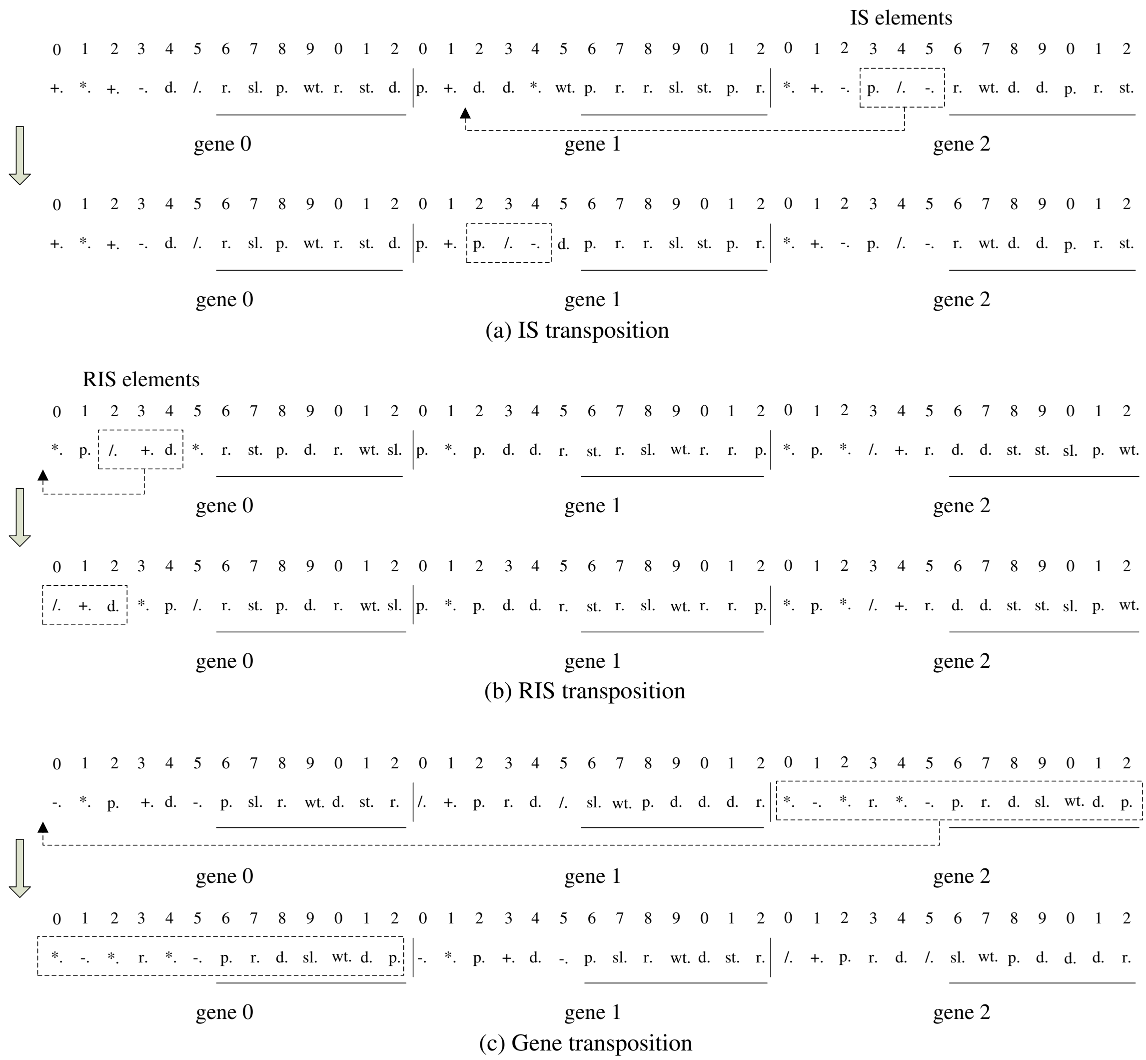


Fig. 4 Transposition of GEP

Table 1 Control parameters settings in GEPSRC and GPSRC

GEPSRC		GPSRC	
Parameters	Value	Parameters	Value
Population size	200	Population size	200
Termination condition	The best solution has not been improved for consecutive 100 evolutionary iterations	Termination condition	The best solution has not been improved for consecutive 100 evolutionary iterations
Maximum length of chromosome	10 for head, 21 for each gene, 3 gene	Maximum length of tree	63
Initialization	Randomly	Initialization	Ramped half-and-half, max. depth of 15
Mutation and transposition	0.03 probability for mutation, and 0.3, 0.1, 0.1 probability for IS, RIS and Gene transposition, respectively. IS elements length is 1, 2, 3, RIS elements length is 1, 2, 3.	Mutation	0.05 probability
Recombination	0.2, 0.5, 0.1 probability for One-point, Two-point and Gene recombination, respectively	Crossover	0.80 probability

rules are selected as benchmarks to which the rules constructed with GEPSRC are compared.

List (list scheduling) Given a set of jobs with release dates, the jobs are ordered in arbitrary list (sequence). Whenever the machine is idle, the first job on the list which is available is scheduled on the machine [13]. This algorithm works in the model with release dates since it does not use the information about processing times of jobs [20].

Modified PSW algorithm The algorithm produces non-preemptive schedules from preemptive ones. Given a set of jobs with release dates and processing times, a preemptive schedule is first formed using the SRPT rule. Under this rule, the machine always picks jobs with the shortest remaining processing time among those already released at the current time and processes these first. Each job will have a (preemptive) completion time C_j. Next, an ordered list L of jobs is formed based on their preemptive completion time C_j using a simple sort. A non-preemptive schedule is then obtained if the first job in L is continued to be assigned to the machine when it is freed and delete it from L. The algorithm yields good solutions for the problem on average [19].

Earliest due date rule All jobs currently waiting processing in the queue of the machine are listed in ascending order of their due dates d_i. The first job in the list is processed next at the machine. This rule is the most popular due-date-based rule. It is known to be used as a benchmark for reducing maximum tardiness and variance of tardiness [42].

Montagne rule Montagne rule (MON) sequences jobs currently waiting processing in ascending order of the following ratio $p_i/(P-d_i)$, where P denotes the sum of the processing time of all jobs [43]. The first job in the list is processed next at the machine. This means that a job with a due date close to the sum of the processing time of all jobs is likely to be scheduled on a later stage. Conversely, jobs with early due dates are given extra priority. MON performs well on different types of single-machine tardiness problems [33].

Minimum slack time rule Minimum slack time rule (MST) lists jobs currently waiting for processing in ascending order of their slack sl_i, where slack for a job is computed by subtracting its processing time at the machine p_i and the current time t from its due date d_i, i.e., $\mathrm{sl}_i=d_i-t-p_i$. The first job in the list is processed next at the machine. This rule is also used to reduce total tardiness of jobs [44].

Modified due date rule The jobs are listed in ascending order of their modified due date md_i, where the modified due date of a job is the maximum of its due date and its remaining processing time, i.e., $\mathrm{md}_i=\max\,(t+p_i,\ d_i)$. This means that once a job becomes critical, its due date becomes its earliest completion time. The first job in the list is processed next at the machine. This rule is aimed to minimize total tardiness of jobs [45].

Cost over time rule When a job is projected to be tardy (i.e., its slack is 0), its priority value reduces to $1/p_i$. On the other hand, if a job is expected to be very early where the slack exceeds an estimation of the delay cost, the priority value for the job increases linearly with decreases in slack. Cost over time rule (COVERT) uses a worst-case estimate of delay as the job processing times multiplied by a look-ahead parameter k. In other words, the priority value of job i is computed as $\frac{1}{p_i}\times\left(1-\frac{(d_i-t-p_i)^+}{k\times p_i}\right)^+$, where $(X)^+=\max(0,\ X)$ [46]. Thus, the priority value of a job increases linearly from 0 when slack is very high to $1/p_i$ when the status of job becomes tardy. The job with the largest COVERT priority value is processed next at the machine.

Apparent tardiness cost rule Apparent tardiness cost rule (ATC), a modification of COVERT, estimates the delay penalty using an exponential discounting formulation, i.e., priority value of job i is computed with $\frac{1}{p_i}\times e^{-\frac{(d_i-t-p_i)^+}{k\times p_i}}$ [47]. If a job is tardy, ATC reduces to $1/p_i$. If the job experiences very high slack, ATC reduces to the MST. It must be noted that if the estimated delay is extremely large, ATC again reduces to $1/p_i$, which is different from COVERT. The job with the largest priority value is processed next at the machine.

In both COVERT and ATC, the look-ahead factor k can significantly affect performance; k is varied from 0.5 to 4.5 in increments of 0.5, and the objective function value where COVERT and ATC each performs best is recorded.

GPRules GP-based scheduling approaches which automatically construct effective rules for a given scheduling environment have been investigated recently, and they have achieved good performance [1, 31, 33-39]. Therefore, besides the classical online heuristics and scheduling rules mentioned above, the rules evolved by GP are also used to evaluate the efficiency of GEPSRC. In the paper, GP-based scheduling rules constructor (GPSRC) is also implemented in which GP is used as the reasoning mechanism to search the SRs space. The description of GPSRC is provided in the Appendix. The control parameters settings for GPSRC are summarized in Table 1, column 2. It is noticeable that an individual of GP is represented as a rooted tree, while an individual of GEP map into several sub-trees which are connected with each other to form a bigger tree as describe in Section 4.2.2. It is unique character of GEP that an individual may consist of more than one gene, which significantly improve the expression ability of the genotype/phenotype. But a maximum program size of 63 was used in both GP and GEP so that comparisons could be made between all the experiments (to be more precise, for GEP with three genes with head length 10, maximum program size of GEP equals 63).

The measure used for heuristic comparison is the percent-relative error computed as

$$\%\text{Error} = \frac{O^k - O^l}{O^l}$$

Where O^k is the average objective value over the test set of problem instances obtain by heuristic k, and O^l is the average objective value over the test set of problem instances obtain by heuristic l. A negative value indicates that heuristic k performs better than heuristic l.

5.3 Design of experiments

In this section, we generate a series of training sets and test sets that represent a set of problem instances of varying operating conditions to evolve rules with GEPSRC and evaluate them.

Problem instances are randomly generated with the instance generation approach used by Jakobovic and Budi [1]. Each scheduling problem instance is defined with the following parameters:

- n the number of jobs. Its value is 10, 50, or 100;
- p_j processing time of job j, j=1,…, n. The values of processing time are assumed as integers and drawn out of U[1,100], U[100, 200], or U[200, 300], where U refers to the uniform distribution;
- r_j release date of job j, j=1,…, n. Release dates are integers chosen randomly from U[0, 1/2 * P], where U refers to the uniform distribution and P denotes the sum of the processing time of all jobs;
- d_j due date of job j, j=1,…, n. Due dates are integers and drawn out of $U\left[r_j + \left(P - r_j\right) * (1 - T - R/2),\ r_j + \left(P - r_j\right) * (1 - T + R/2)\right]$, where U refers to the uniform distribution, P denotes the sum of the processing time of all jobs, T is due date tightness factor which represents the expected percentage of late jobs, and R is due date range factor which defines the dispersion of the due dates values. T and R are assigned values of 0.1, 0.5, or 0.9.
- w_j weight of job j, j=1,…, n. We assume all jobs relatively equal, i.e., w_j=1.

Table 2 summarizes the different values of the parameters used to generate problem instances of varying operating conditions.

Table 2 Simulation parameters setting

Parameter	Levels	Values
Number of jobs (n)	Small (S)	10
	Moderate (M)	50
	Large (L)	100
Processing time of jobs (p)	Small (S)	U[1,100]
	Moderate (M)	U[100,200]
	Large (L)	U[200,300]
Due date tightness (T)	Loose (L)	0.1
	Moderate (M)	0.5
	Tight (T)	0.9
Due date range (R)	Small (S)	0.1
	Moderate (M)	0.5
	Large (L)	0.9

Eighteen training sets are generated to construct rules for a given performance measure. In the first nine training sets, the value of n and p of each training set was fixed, whereas T and R assume values of 0.1, 0.5, and 0.9 in various combinations (3×3=9). In the remaining nine training sets, the value of T and R in each training set was fixed, whereas n and p assume value of 10, 50, 100 and U[1,100], U[100, 200], or U[200, 300] in various combinations (3×3=9). The number of the instances generated for each of nine combinations of parameter in each training set (called *sample size*) is noticeable because the composition of the training sets can significantly influence the generality of the evolved SRs. Extensive experiments were conducted to investigate the impact of *sample size* on the success of learning. The most appropriate *sample size* for this research was determined to be three, and results showed that a large *sample size* was unbeneficial to construct effective SRs that generalize well to unseen scheduling instances in test sets. Therefore, a training set that consisted of 27 problem instances is used to construct SRs in each individual GEPSRC run. Five runs were conducted in total for each training set. In addition, 18 different test sets of the similar composition using the same parameters are generated for evaluation purposes.

5.4 Analysis of results

Various experiments are conducted to evaluate the efficiency of the proposed GEP-based scheduling approach GEPSRC in the comparison with the benchmark heuristics listed in Section 5.2.

5.4.1 Minimizing makespan

Table 3 shows the makespan results of benchmark heuristics presented in Section 5.2 and rules evolved

Table 3 Comparison of benchmark heuristics and GEPRules relative to List for minimizing makespan problem on test sets #1–9

Test set #	n	p	List % Error	MPSW %Error	EDD %Error	MON %Error	MST %Error	MDD %Error	COVERT %Error	ATC %Error	GPRules			GEPRules		
											Ave % Error	Best % Error	Worst %Error	Ave % Error	Best % Error	Worst %Error
1	S	S	0.000	7.375	5.347	7.003	5.331	6.445	26.450	26.450	0.000	0.000	0.000	0.000	0.000	0.000
2	S	M	0.000	0.713	5.535	2.177	6.202	5.139	12.968	12.968	0.068	0.000	0.338	0.000	0.000	0.000
3	S	L	0.000	0.401	4.970	3.435	6.206	4.166	12.975	12.975	0.000	0.000	0.000	0.000	0.000	0.000
4	M	S	0.000	8.120	2.217	5.346	1.765	2.187	25.085	32.006	0.001	0.000	0.007	0.000	0.000	0.000
5	M	M	0.000	0.493	3.780	2.277	3.658	3.376	9.217	9.682	0.000	0.000	0.000	0.000	0.000	0.000
6	M	L	0.000	0.170	4.036	1.782	4.276	3.650	9.012	9.464	0.000	0.000	0.000	0.000	0.000	0.000
7	L	S	0.000	7.026	1.049	4.586	0.949	1.038	9.568	11.354	0.000	0.000	0.000	0.000	0.000	0.000
8	L	M	0.000	0.423	3.149	1.613	2.890	2.707	30.798	31.668	0.000	0.000	0.000	0.000	0.000	0.000
9	L	L	0.000	0.112	3.686	1.704	3.326	3.134	16.392	16.392	0.000	0.000	0.000	0.000	0.000	0.000

by GEPSRC (GEPRules) on different test sets. “Ave” column is the average performance of the GEPRules or GPRules over the test set from the five run. The “best” and “worst” columns summarize the performance of the best and the worst performing rules over the five runs, respectively.

The results indicate that the list scheduling emerges as the best among the benchmark heuristics in minimizing makespan objective in all cases regardless of the changing the number and processing time of jobs. Modified PSW (MPSW) algorithm also exhibits notable performance when jobs' processing times are moderate or large, but its performance degrades when jobs' processing times are small. Among the heuristics, the Covert and ATC perform the worst in minimizing makespan. This is because these two rules concentrate significantly on due dates. The obtained scheduling minimizes the objectives related to due date, such as tardiness, but the completion time of the whole schedule is ignored.

For all cases, GEPRules exhibit the best performance as list scheduling. GPRules perform slightly worse than GEPRules in several cases. The average percent error of GPRules relative to List scheduling ranged from 0.001% to 0.068% and worst percent error of GPRules relative to list scheduling ranged from 0.007% to 0.338%.

The best rule learned by GEPSRC for test set #1 is formulized as r, i.e., it contains only the release date information, which indicates that the release dates information is valuable and that the due dates information is irrelevant to minimizing makespan problem. The phenomenon also exists in other GEPRules discovered for other scenarios. Therefore, it is inferred that the release date information contributes mainly in reducing makespan objective and that GEPSRC is capable of identifying the significance of job release dates for the performance measure, although other attributes of jobs are also provided to it. On the other hand, the rules discovered by GP are more complex and cannot explicitly interpret the relationship among the attributes of job.

Table 4 Comparison of benchmark heuristics and GEPRules relative to MPSW for minimizing flow time problem on test sets #1–9

Test set #	n	p	List % Error	MPSW %Error	EDD % Error	MON % Error	MST % Error	MDD % Error	COVERT %Error	ATC % Error	GPRules			GEPRules		
											Ave % Error	Best % Error	Worst %Error	Ave % Error	Best % Error	Worst %Error
1	S	S	22.981	0.000	34.606	10.451	46.505	21.057	43.954	42.795	−8.004	−8.643	−7.566	−8.759	−8.802	−8.744
2	S	M	9.477	0.000	24.680	8.804	29.589	19.705	37.622	40.290	−0.713	−0.852	−0.546	−0.759	−0.878	−0.714
3	S	L	5.145	0.000	18.082	11.471	22.341	13.371	34.593	39.501	−0.529	−0.576	−0.493	−0.552	−0.576	−0.499
4	M	S	29.951	0.000	40.434	6.357	45.162	24.656	30.319	31.706	−18.160	−18.184	−18.113	−18.202	−18.218	−18.186
5	M	M	15.976	0.000	29.769	15.585	29.985	23.171	29.781	30.392	−1.122	−1.144	−1.103	−1.128	−1.144	−1.109
6	M	L	9.624	0.000	23.736	12.495	24.853	19.504	25.082	26.380	−0.412	−0.414	−0.411	−0.412	−0.414	−0.411
7	L	S	35.573	0.000	37.362	7.349	41.237	24.841	27.313	26.374	−17.399	−17.414	−17.377	−17.356	−17.408	−17.278
8	L	M	17.287	0.000	28.827	14.484	28.208	22.176	24.846	27.561	−1.068	−1.083	−1.063	−1.057	−1.065	−1.054
9	L	L	10.167	0.000	22.641	12.696	21.578	17.751	20.695	21.898	−0.246	−0.265	−0.197	−0.255	−0.255	−0.255

5.4.2 Minimizing flow time

Table 4 shows the flow time results of benchmark heuristics and GEPRules on different test sets. Since MPSW algorithm produces good solutions for the flow time problem, the average objective function values obtained from the other heuristics, including the rules that are discovered by GEPSRC and GPSRC, are compared to those values form MPSW. The second rank goes to MON rule, but MON performs worse than list scheduling in the cases where the jobs' processing time is large. List scheduling exhibits better performance than modified due date rule (MDD) expect for the cases where the jobs' processing time is small. The results in Table 4 indicate that MST obtains the worst result when jobs' processing times are small, however, its performance increase as the processing time increase.

Table 4 also shows that the rules constructed by GEPSRC and GPSRC outperform MPSW algorithm in all cases, especially in the cases where the jobs' processing time is small. In the comparison with GPRules, GEPRules shows better performance when the number of jobs are small or moderate. The improvement over MPSW obtained from GEPSRC is more than that obtained from GPSRC by 0.755% for the average percent error, 0.159% for the best percent error, and 1.178% for the worst percent error. However, GEPSRC's performance degrades when the number of jobs becomes large, but these result in small degradation in average objective function value. The improvement over MPSW obtained from GEPSRC is less than that obtained from GPSRC by 0.043% for the average percent error, 0.018% for the best percent error, and 0.099% for the worst percent error.

GEPSRC exhibits the ability to intelligently select the useful attributes from candidate ones to automatically construct effective SRs. Take the best rules constructed by GEPSRC for test set #4 and #9 (Rule-F4 and Rule-F9) for example.

$$p + \mathrm{wt}^2 + \frac{r \cdot \mathrm{wt} - \mathrm{wt}^2}{p} \qquad (\text{Rule} - \text{F4})$$

$$p + 2\mathrm{wt} + \frac{r}{\mathrm{wt}} \qquad (\text{Rule} - \text{F9})$$

From Rule-F4 and Rule-F9, it is easy to find that the specific due date parameters of due date d and slack sl is not relevant to the criterion of minimizing flow time, regardless of the variation of due dates of the jobs to be scheduled, whereas release date r and processing time p help to reduce the flow time of the jobs (recall that from the definition of waiting time wt in Section 4.2.1; waiting time wt and release date r are correlated). Moreover, when all jobs are available simultaneously, the rule above may be reduced into p, i.e., SPT rule, which produces optimal solution for the special case [14].

As for the rules discovered by GPSRC, the relationship among the attributes of jobs cannot be explicitly explained from their expressions since they are usually quite complex.

5.4.3 Minimizing maximum lateness

Table 5 shows the performance of the heuristics for the criterion of maximum lateness. Earliest due date rule (EDD) performs well in these test sets. For this reason, the average objective function values obtained from the other heuristics are compared to those values from EDD. The results show that MON performs better than MST when jobs' processing times are moderate or large, but its performance degrades when jobs' processing times are small. List scheduling and MPSW algorithm obtain good results in minimizing makespan and flow time problems, respectively (see Sections 5.4.1

Table 5 Comparison of benchmark heuristics and GEPRules to EDD for minimizing maximum lateness problem on test sets #1–9

Test set #	n	p	List % Error	MPSW %Error	EDD %Error	MON %Error	MST %Error	MDD %Error	COVERT %Error	ATC % Error	GPRules			GEPRules		
											Ave % Error	Best % Error	Worst %Error	Ave % Error	Best % Error	Worst %Error
1	S	S	27.873	87.440	0.000	39.645	18.138	49.521	164.447	164.447	−16.164	−16.909	−14.655	−16.583	−16.909	−15.315
2	S	M	20.665	55.784	0.000	9.999	23.108	35.034	108.054	108.054	−18.806	−18.936	−18.700	−18.825	−18.936	−18.798
3	S	L	29.528	83.658	0.000	9.870	25.317	47.507	148.285	148.306	−12.396	−14.581	−6.573	−14.487	−14.641	−14.052
4	M	S	86.639	231.442	0.000	69.127	38.879	68.811	232.310	266.956	−10.468	−10.570	−10.371	−10.437	−10.570	−10.252
5	M	M	81.048	183.409	0.000	34.855	47.752	77.370	205.405	209.058	−17.272	−17.311	−17.211	−17.285	−17.315	−17.268
6	M	L	73.449	187.331	0.000	20.858	46.203	84.471	175.690	183.151	−19.864	−20.011	−19.619	−20.011	−20.011	−20.011
7	L	S	111.014	288.834	0.000	77.912	46.460	102.376	191.295	199.247	−5.526	−5.526	−5.526	−5.524	−5.526	−5.516
8	L	M	97.332	221.867	0.000	34.879	46.868	105.913	262.961	286.395	−16.257	−16.315	−16.216	−16.215	−16.275	−16.153
9	L	L	84.010	202.549	0.000	22.617	43.973	102.620	251.479	253.637	−18.917	−18.943	−18.892	−18.923	−18.929	−18.900

and 5.4.2). However, they perform poorly in minimizing maximum lateness in comparison with EDD. This is because the due dates of jobs are ignored by the two algorithms. COVERT and ATC perform worst among the heuristics for the due-date-related objective, although they are also due-date-based rules. The reason is that they try to minimize the deviation between the completion time and due date for each job, which may degrade the objective of minimizing maximum lateness.

From the results, it is also easy to found that both GEPRules and GPRules exhibit much better behave than EDD. When the number of jobs is small and moderate, GEPSRC exhibits better performance than GPSRC, except on the test case #4. However, the performance of GEPSRC is slightly worse than GPSRC when the number of jobs is large. The improvement on average performance obtained from GEPSRC over EDD is less than that obtained from GPSRC by 0.042%, the improvement on best learned rule performance obtained from GEPSRC is less than that obtained from GPSRC by 0.04%, and the improvement on worst learned rule performance obtained from GEPSRC is less than that obtained from GPSRC by 0.119%.

Take a closer look at the best rules constructed by GEPSRC for test set #5 and #9 (Rule-L5 and Rule-L9).

$$d + p \cdot \mathrm{wt}^2 + \frac{r}{\mathrm{wt}} \qquad \text{(Rule} - \text{L5)}$$

$$d + \mathrm{sl} + r \cdot \mathrm{wt} \qquad \text{(Rule} - \text{L9)}$$

The learned rules show the visible role of due date-related attribute d and sl. This seems logical, as the objective function is due-date-related and is aligned with the general conclusions of the scheduling research community. Moreover, when all jobs are available simultaneously, the rules above may be reduced into d, i.e., EDD rule or the combination rule of EDD and MST. The rules discovered by GPSRC can not explicitly explain the relationship among the attributes of jobs.

Further experiments are conducted on test sets #10–18 to evaluate the performance of heuristics under a variety of level of due date tightness factor and range factor. The results are summarized in Table 6. Since EDD performs well in these test sets, the average objective function values obtained from the other rules are compared to those values from EDD. However, EDD does not guarantee the best solutions for the maximum lateness problem, shown by other rules receiving a negative percent error score.

All rules seem to perform better under the small due date range conditions than under the large due date range conditions. Under the small due date range conditions, list scheduling performs better EDD in minimizing maximum lateness. However, its performance degrades significantly

Table 6 Comparison of benchmark heuristics and GEPRules relative to EDD for minimizing maximum lateness on test set #10–18

Test set #	T	R	List % Error	MPSW % Error	EDD % Error	MON % Error	MST % Error	MDD % Error	COVERT % Error	ATC % Error	GPRules			GEPRules		
											Ave % Error	Best % Error	Worst % Error	Ave % Error	Best % Error	Worst % Error
10	L	S	−19.656	28.143	0.000	−13.849	32.730	6.105	139.826	197.562	−39.313	−39.313	−39.313	−39.313	−39.313	−39.313
11	L	M	590.547	797.819	0.000	425.494	107.261	5.233	892.131	963.855	−129.871	−132.884	−124.351	−131.443	−132.884	−127.862
12	L	L	254.368	364.452	0.000	213.818	7.543	0.000	332.256	309.488	−28.503	−29.083	−26.839	−28.079	−28.493	−27.474
13	M	S	−2.275	57.907	0.000	10.549	8.831	20.171	87.623	87.623	−5.236	−5.236	−5.236	−5.150	−5.236	−4.805
14	M	M	42.819	126.785	0.000	3.619	42.971	43.799	146.852	145.050	−14.484	−14.484	−14.484	−14.484	−14.484	−14.484
15	M	L	173.713	287.080	0.000	9.699	108.399	88.777	225.492	225.492	−18.795	−18.985	−18.058	−19.791	−23.047	−18.963
16	T	S	−0.441	62.285	0.000	14.219	2.102	54.208	75.435	75.435	−2.027	−2.027	−2.027	−1.979	−2.027	−1.799
17	T	M	18.070	82.071	0.000	15.240	21.844	73.579	90.233	90.233	−6.449	−6.532	−6.241	−6.483	−6.532	−6.410
18	T	L	59.455	148.061	0.000	25.140	61.407	125.245	150.433	148.409	−6.257	−6.266	−6.232	−6.250	−6.262	−6.232

with the increase of due date range. Both MON and MDD outperforms MPSW under all cases. As expected, COVERT and ATC perform well when due date are tight, but they still perform poor compared with other heuristics.

Under all conditions, GEPRules and GPRules perform much better than all the benchmark heuristics. In most cases, GEPRules perform the best or tie for best. However, under larger due date range condition, the performance of GEPRules degrades slightly, but it results in small degradation in the average objection values obtained by GEPRule. In the cases where the GEPRules perform worse than GPRules, the percent error improvement relative to EDD obtained by GEPRule is less than that obtained by GPRule by 0.424% for average percent error, 0.59% for best percent error, and 0.431% for worst percent error. On the cases where the GEPRules perform better than GPRules, the percent error improvement relative to EDD obtained by GEPRules is more than that obtained by GPRules by 1.572% for average percent error, 4.062% for best percent error, and 3.511% for worst percent error.

Take a closer look at the best rules constructed by GEPSRC for test set 11# and 17# (Rule-L11 and Rule-L17).

$$2d + \mathrm{sl} + p + r \cdot \mathrm{wt} \qquad (\mathrm{Rule} - \mathrm{L11})$$

$$d + d^2 \cdot \mathrm{wt} + \mathrm{wt} \qquad (\mathrm{Rule} - \mathrm{L17})$$

The learned rules show that under the loose due date tightness and moderate due date range condition, i.e., on the test set 11#, in order to minimize the maximum lateness, the parameters of d, sl, p, r, and wt all contribute to the success of the scheduling. Whereas, under the tight due date tightness and moderate due date range condition, i.e., on the test set 17#, the d and wt play dominating rule on the scheduling decision in order to minimizing the maximum lateness. It means that the GEPSRC can identify the characteristics of the operation conditions and construct appropriate rules.

5.4.4 Minimizing tardiness

Table 7 summarizes the performance of the heuristics and rules discovered by GEPSRC and GPSRC relative to the minimizing the tardiness problem. MDD produces good approximate solutions for the scheduling problem and, hence, is used as a benchmark in this experiment. MON also produces good approximate solutions and ranks the second, except in several cases MON performs worse than COVERT and ATC. Although COVERT and ATC behave poor when the number and processing time of jobs are small, the performance of COVERT and ATC increase significantly in the cases where the number and processing time of jobs become large for minimizing the total of tardiness problem. Table 7 also shows that list scheduling, MPSW algorithm, and EDD rule exhibit similar performance, with MPSW outperforming list scheduling and EDD for the cases where the number of jobs is small and EDD outperforming list scheduling and MPSW for the cases where the number of jobs is moderate or large.

Both GEPRules and GPRules perform better than all the benchmark algorithms in all the test sets. From the average percent error, best percent error, and worst percent error, it is easy to infer that GEPSRC performs distinctly more effectively and steadily than GPSRC with regard to the criterion of minimizing tardiness. There are just a few cases where the discovered rule by GEPSRC performs worse than that of GPSRC with the slight performance degradation.

Table 7 Comparison of benchmark heuristics and GEPRules to MDD for minimizing tardiness problem on test sets #1–9

Test set #	n	p	List % Error	MPSW %Error	EDD % Error	MON % Error	MST % Error	MDD % Error	COVERT %Error	ATC % Error	GPRules			GEPRules		
											Ave % Error	Best % Error	Worst % Error	Ave % Error	Best % Error	Worst %Error
1	S	S	32.905	21.195	32.596	15.579	38.458	0.000	47.613	47.427	−11.068	−11.836	−8.687	−11.760	−11.836	−11.604
2	S	M	8.261	2.427	12.349	−0.316	16.878	0.000	36.301	37.037	−10.926	−12.094	−7.038	−11.951	−12.094	−11.791
3	S	L	10.613	8.596	13.142	2.090	10.978	0.000	45.246	48.701	−7.320	−7.853	−5.191	−7.853	−7.853	−7.853
4	M	S	78.086	60.877	57.193	31.009	55.637	0.000	15.351	18.593	−7.951	−8.183	−7.693	−8.029	−8.034	−8.012
5	M	M	28.703	26.888	20.247	6.646	16.441	0.000	13.957	14.244	−8.767	−8.829	−8.694	−8.797	−8.944	−8.574
6	M	L	21.236	26.891	13.058	1.871	11.303	0.000	12.297	13.241	−9.060	−9.074	−9.038	−9.039	−9.071	−9.031
7	L	S	82.816	66.504	48.147	36.350	50.712	0.000	4.262	4.210	−3.891	−3.995	−3.475	−3.974	−3.995	−3.889
8	L	M	33.564	35.504	21.794	6.445	17.071	0.000	7.478	12.713	−6.774	−6.859	−6.639	−6.742	−6.859	−6.634
9	L	L	24.347	31.903	15.712	2.238	9.697	0.000	7.394	8.393	−7.156	−7.168	−7.139	−7.157	−7.185	−7.113

The best rule evolved by GEPSRC for test sets 3# and 7# (Rule-T3 and Rule-T7) is shown below.

$$\mathrm{sl} + 2p + r \cdot \mathrm{wt} \cdot p \qquad (\text{Rule} - \text{T3})$$

$$\mathrm{sl} + p + \mathrm{wt} + \mathrm{wt}^2 \qquad (\text{Rule} - \text{T7})$$

From the formula of Rule-T3 and Rule-T7, it is found that the GEPSRC picks the sl as an indispensable element to construct SRs for the scheduling decision relative to the criterion of tardiness. Essentially, the first term of sl works as MST rule, which performs well under the performance measure of tardiness. Besides sl, it is noticeable that p and r also play an important role for the scheduling decision. The rules discovered by GPSRC cannot explicitly explain the relationship among the attributes of jobs.

Further experiments are conducted on test sets 10–18# to evaluate the performance of the heuristics under a variety of level of due date tightness factor and range factor. The results are summarized in Table 8. MDD performs well in these test sets, and the average objective function values obtained from the other heuristics are compared to those values from MDD.

As expected, under tight due date conditions, COVERT and ATC perform well. But their performance degrades as due date loosen. In most cases, MON performs better than EDD and MST. EDD and MST exhibit similar performance, with EDD outperforming MST for the cases where the due date tight is loose and moderate. The results in Table 8 also indicates that the performance of list scheduling and MPSW algorithm degrades as the due date range become large. This is because list scheduling and MPSW focus only on the release date and processing time of jobs and ignore the due date information so as not to be sensitive to the variability of due date.

The GEPRules and GPRules perform better than all the benchmark algorithms in all the test sets. On all test sets, GEPSRC consistently find more high-performing rules than GPSRC regardless of the variety of due date tight and range. What is more, GEPSRC exhibits the ability to recognize the different operating conditions and to employ the appropriate elements to construct rules in appropriate algebraic combination. Take a closer look at the best rules constructed by GEPSRC for test sets 16# and 18# (Rule-L16 and Rule-L18).

$$\mathrm{sl} + 3p + d \cdot \mathrm{wt} \qquad (\text{Rule} - \text{L16})$$

$$\mathrm{sl} + p + 2\mathrm{wt}^2 \qquad (\text{Rule} - \text{L18})$$

From the learned rules, it is found that although the two rules employ similar elements, they are constructed in

Table 8 Comparison of benchmark heuristics and GEPRules relative to MDD for minimizing tardiness problem on test sets #10–18

Test set #	T	R	List % Error	MPSW % Error	EDD % Error	MON % Error	MST % Error	MDD % Error	COVERT % Error	ATC % Error	GPRules			GEPRules		
											Ave % Error	Best % Error	Worst % Error	Ave % Error	Best % Error	Worst % Error
10	L	S	−23.740	29.695	13.783	−22.644	87.767	0.000	78.314	133.909	−60.553	−60.553	−60.553	−60.553	−60.553	−60.553
11	L	M	1628.782	2300.699	6.396	1337.224	172.511	0.000	1021.571	1558.516	−47.532	−50.195	−46.481	−47.843	−50.481	−47.302
12	L	L	5421.792	8509.184	0.000	4795.660	89.907	0.000	2215.397	2139.884	−70.235	−78.831	−51.397	−74.681	−78.831	−73.644
13	M	S	15.629	50.797	25.551	16.256	34.066	0.000	17.875	17.115	−10.115	−10.764	−9.541	−10.646	−10.764	−10.565
14	M	M	68.378	102.077	22.455	9.002	32.060	0.000	23.655	22.182	−25.429	−25.785	−24.969	−25.789	−25.827	−25.776
15	M	L	271.801	277.604	21.845	0.923	33.848	0.000	34.108	33.262	−37.526	−39.225	−35.867	−38.599	−39.225	−38.124
16	T	S	23.832	9.250	27.777	8.933	29.542	0.000	6.840	6.764	−3.806	−3.865	−3.724	−3.828	−3.902	−3.825
17	T	M	27.141	13.119	28.750	9.746	24.981	0.000	5.841	5.779	−3.987	−4.016	−3.936	−4.001	−4.022	−3.980
18	T	L	27.153	17.819	27.285	7.570	19.315	0.000	5.726	5.506	−3.254	−3.361	−3.058	−3.298	−3.369	−3.268

different algebraic combinations of these elements according to the operation conditions.

5.4.5 Computational requirement

GEPSRC and GPSRC are both implemented in C++. The experiments perform on a PC (Windows XP, CPU 2.00 GHz, Memory 2.00 GB). Table 9 summarized the CPU times required to train the GEP and GP on each training set of 27 randomly generated problem instances for the minimizing tardiness problem. It is easy to find that there is significant difference between the CPU times required to construct the SRs for the performance criterion. The CPU time required to train GP is 1–10 times more than that required to train GEP. Relative to other performance measures, there is also significant difference between the CPU time required to train GEP and GP.

6 Conclusion and future work

This paper considered the DSMSPs with job release dates and proposed the GEPSRC to automatically evolve SRs for the problems. GEP works as heuristic search to search the space of algorithm. For minimizing makespan, total flow time, maximum lateness, and total tardiness problem, the performance of GEPSRC was evaluated on extensive randomly generated test cases. SRs obtained from GEPSRC performed more effectively and steadily than those obtained from GPSRC and prominent heuristics selected for literature. Moreover, SRs obtained from GEPSRC can be expressed simply and explicable in some way.

A traditional GEP framework was used in this paper. Since GEP was proposed in 2001, the research on GEP has been developing promptly. Many exciting fruits have been reported in literature recently, which may contribute to improving the ability for GEP to construct more effective SRs. Promising research work may include: (1) introduce other mechanisms or techniques such as immunity mechanism or transfer gene technique into GEP's framework to increase its speed of convergence; (2) add other potential functions, such as relational functions, logical functions, or conditional functions, into function set to express the SRs more effectively; (3) design a special analyzer embodied in GEP to evolve SRs which are easier to analyze the effect of various attributes of jobs on the scheduling decision qualitatively and quantitatively. To extend the work on dynamic single-machine scheduling problems to job shop environment is also part of the future work.

Acknowledgements This research was supported by the 863 project of China under Grant No. 2006AA04Z131, the New Century Excellent Talents in University under Grant No. NCET-08-0232 and the National Natural Science Foundation of China under Grant No. 50825503.

Appendix

The framework of GPSRC is similar with that of GEPSRC which is described in Section 4.1. However, the difference between them is that the former uses GP as the evolutionary learning mechanism. Genetic Programming belongs to the family of evolutionary computation methods, invented by Cramer [48] and further developed by Koza [49]. GP combines efficiently the concepts of evolutionary computation and automatic programming [33].

A potential solution of an optimization problem is appropriately coded with elements from FS and TS into an individual, i.e., a rooted tree, and a population of these tree structures is employed for the evolution of optimal or near optimal solutions through successive generations.

The general procedure of GP algorithm can be viewed as a four-step cycle [31]:

Step 1: An initial population of individuals is created with the method of ramped half-and-half
Step 2: Each individual in the population is then decoded so that its performance (fitness) can be evaluated
Step 3: A selection mechanism is used to choose a subset of individuals according with these fitness values

Table 9 Average CPU time per individual run for training GEP and GP on each training set relative to minimizing tardiness problem

Test set #	Average CPU time (s)		Test set #	Average CPU time (s)		Test set #	Average CPU time (s)	
	GEP	GP		GEP	GP		GEP	GP
1	8.8876	51.7092	7	342.791	1465.93	13	152.888	1012.12
2	5.9658	31.884	8	565.29	1026.51	14	142.794	437.81
3	4.6534	24.3876	9	556.853	1089.98	15	172.003	825.988
4	79.3	461.497	10	101.166	599.157	16	179.55	1081.51
5	98.6372	458.712	11	123.472	224.425	17	166.528	1892.66
6	88.7002	311.706	12	105.019	293.075	18	193.603	553.297

Step 4: These individuals either survive intact to the new population, or they are genetically modified through a number of operators. If the terminal condition is satisfied, the procedure is finished, those individuals who perform the best (i.e., are the most fit) are the solution of the optimization problem; otherwise, turn to Step 2.

Crossover and mutation are the two major operators that are applied for the genetic modification of individuals.

Crossover Crossover begins by choosing two trees from the current population according to their fitness probabilistically. A sub-tree in each parent individual is selected at random. The randomly chosen sub-trees are then swapped, creating two new individual trees.

Mutation The mutation operation involves randomly selecting a sub-tree within a parent individual that has been selected from the population based on its fitness and replacing it with a randomly generated sub-tree. The generated sub-tree is created by randomly selecting elements from FS and TS.

The brief descriptions serve only to provide background information. For more detailed discussions of GP, the reader in encouraged to refer to [31, 48, 49].

References

1. Jakobovic D, Budin L (2006) Dynamic scheduling with genetic programming. Lect Notes Comput Sci 3905:73–84
2. Holthaus O, Rajendran C (1997) New dispatching rules for scheduling in a job shop - an experimental study. Int J Adv Manuf Technol 13(2):148–153
3. Kianfar K, Ghomi SMT, Karimi B (2009) New dispatching rules to minimize rejection and tardiness costs in a dynamic flexible flow shop. Int J Adv Manuf Technol 45(7–8):759–771
4. Wang JB, Wang LY, Wang D, Huang X, Wang XR (2009) A note on single-machine total completion time problem with general deteriorating function. Int J Adv Manuf Technol 44(11–12):1213–1218
5. Ham M, Fowler JW (2008) Scheduling of wet etch and furnace operations with next arrival control heuristic. Int J Adv Manuf Technol 38(9–10):1006–1017
6. Norman B, Bean J (1999) A genetic algorithm methodology for complex scheduling problems. Nav Res Logist 46:199–211
7. Panneerselvam R (2006) Simple heuristic to minimize total tardiness in a single machine scheduling problem. Int J Adv Manuf Technol 30(7–8):722–726
8. Haupt R (1989) A survey of priority rule-based scheduling. OR Spektrum 11:3–16
9. Catoni O (1998) Solving scheduling problems by simulated annealing. SIAM J Control Optim 36(5):1639–1675
10. Finke DA, Medeiros DJ, Traband MT (2002) Shop scheduling using Tabu search and simulation. In: Yucesan E, Chen CH, Snowdon JL, Charnes JM (eds) Proceedings of the 2002 winter simulation conference, 8–11 December 2002 San Diego. WSC, New York, pp 1013–1017
11. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison–Wesley, Boston
12. Aytug H, Lawley MA, Mckay K, Mohan S, Uzsoy R (2005) Executing production schedules in the face of uncertainties: a review and some future directions. Eur J Oper Res 161(1):86–110
13. Graham RL (1966) Bounds for certain multiprocessor anomalies. Bell Syst Tech J 45:1563–1581
14. Smith WE (1956) Various optimizers for single-stage production. Nav Res Logist Q 3:59–66
15. Baker KR (1974) Introduction to sequencing and scheduling. Wiley, New York
16. Hoogeveen JA, Vestjens APA (1996) Optimal on-line algorithms for single-machine scheduling. Lect Notes Comput Sci 1084:404–414
17. Phillips C, Stein C, Wein J (1995) Minimizing average completion time in the presence of release dates. Math Program 82:199–223
18. Kellerer H, Tautenhahn T, Woeginger GJ (1999) Approximability and nonapproximability results for minimizing total flow time on a single machine. SIAM J Comput 28(4):1155–1166
19. Guo Y, Lim A, Rodriguesc B, Yu S (2004) Minimizing total flow time in single machine environment with release time: an experimental analysis. Comput Ind Eng 47:123–140
20. Sgall J (1998) On-line scheduling. Lect Notes Comput Sci 442:196–231
21. Pruhs K, Sgall J, Torng E (2004) Online scheduling. In: Leung JYT (ed) Handbook of scheduling: algorithms, models and performance analysis. Chapman & Hall/CRC, Boca Raton
22. Blackstone JH, Phillips DT, Hogg GL (1982) A state-of-the-art survey of dispatching rules for manufacturing job shop operations. Int J Prod Res 20(1):27–45
23. Jeong KC, Kim YD (1998) A real-time scheduling mechanism for a flexible manufacturing system using simulation and dispatching rules. Int J Prod Res 36(9):2609–2626
24. Yin YL, Rau H (2006) Dynamic selection of sequencing rules for a class-based unit-load automated storage and retrieval system. Int J Adv Manuf Technol 29(11–12):1259–1266
25. Chen CC, Yih Y (1996) Identifying attributes for knowledge-base development in dynamic scheduling environments. Int J Prod Res 34(6):1739–1755
26. El-Bouri A, Shah P (2006) A neural network for dispatching rule selection in a job shop. Int J Adv Manuf Technol 31(3–4):342–349
27. Aytug H, Koehler GJ, Snowdon JL (1994) Genetic learning of dynamic scheduling within a simulation environment. Comput Oper Res 21(8):909–925
28. Trappey AJC, Lin GYP, Ku CC, Ho PS (2007) Design and analysis of a rule-based knowledge system supporting intelligent dispatching and its application in the TFT-LCD industry. Int J Adv Manuf Technol 35(3–4):385–393
29. Singh A, Mehta NK, Jain PK (2007) Multicriteria dynamic scheduling by swapping of dispatching rules. Int J Adv Manuf Technol 34(9–10):988–1007
30. Yang HB, Yan HS (2009) An adaptive approach to dynamic scheduling in knowledgeable manufacturing cell. Int J Adv Manuf Technol 42(3–4):312–320
31. Geiger CD, Uzsoy R, Aytug H (2006) Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. J Sched 9:7–34
32. Koza JR (2007) Introduction to genetic programming. In: Lipson H (ed) Proceedings of GECCO 2007: genetic and evolutionary computation conference, 7-11 July 2007 London. ACM, London, pp 3323–3365
33. Dimopoulos C, Zalzala AMS (2001) Investigating the use of genetic programming for a classic one-machine scheduling problem. Adv Eng Softw 32(6):489–498

34. Yin WJ, Liu M, Wu C (2003) Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In: Sarker R et al (eds) Proceeding of CEC2003: congress on evolutionary computation, 9-12 December 2003 Canberra, Australia. IEEE, Piscataway, pp 1050–1055
35. Atlan L, Bonnet J, Naillon M (1994) Learning distributed reactive strategies by genetic programming for the general job shop problem. In: Dankel D, Stewan J (eds) Proceedings of The 7th annual Florida artificial intelligence research symposium, 5-6 May 1994 Pensacola Beach, Florida, USA. IEEE, Piscataway
36. Miyashita K (2000) Job-shop scheduling with genetic programming. In: Whitley LD, Goldberg DE et al (eds) Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2000), 8-12 July 2000 Las Vegas, Nevada, USA. Kaufmann, San Francisco, pp 505–512
37. Ho NB, Tay JC (2003) Evolving dispatching rules for solving the flexible job shop problem. In: Corne D (ed) Proceedings of the 2005 IEEE congress on evolutionary computation, 2–4 September 2005 Edinburgh, UK. IEEE, Piscataway, pp 2848–2855
38. Tay JC, Ho NB (2007) Designing dispatching rules to minimize total tardiness. Stud Comput Intel 49:101–124
39. Tay JC, Ho NB (2008) Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problem. Comput Ind Eng 54(3):453–473
40. Ferreira C (2001) Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst 13(2):87–129
41. Hardy Y, Steeb WH (2002) Gene expression programming and one-dimensional chaotic maps. Int J Mod Phys C 13(1):13–24
42. Jayamohan MS, Rajendran C (2000) New dispatching rules for shop scheduling: a step forward. Int J Prod Res 38:563–586
43. Montagne ER (1969) Sequencing with time delay costs. Industrial Engineering Research Bulletin, Arizona State University 5
44. Oliver H, Chandrasekharan R (1997) Efficient dispatching rules for scheduling in a job shop. Int J Prod Econ 48(1):87–105
45. Kanet JJ, Li XM (2004) A weighted modified due date rule for sequencing to minimize weighted tardiness. J Sched 7(4):261–276
46. Bhaskaran K, Pinedo M (1992) Dispatching. In: Salvendy G (ed) Handbook of industrial engineering. Wiley, New York, pp 2184–2198
47. Vepsalainen APJ, Morton TE (1987) Priority rules for job shops with weighted tardiness costs. Manage Sci 33:1035–1047
48. Cramer NL (1985) A representation for the adaptive generation of simple sequential programs. In: Grefenstette JJ (ed) Proceedings of The First International Conference on Genetic Algorithms and their Applications, 24-26 July 1985 Pittsburgh, USA. Erlbaum, Hillsdale, pp 183–187
49. Koza JR (1994) Genetic programming II: automatic discovery of reusable programs. MIT, Cambridge