2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Sequential Genetic Programming Method to Learn Forward Construction Heuristics for Order Acceptance and Scheduling

Su Nguyen, Mengjie Zhang and Mark Johnston
Evolutionary Computation Research Group
Victoria University of Wellington, PO Box 600, Wellington, New Zealand
{su.nguyen,mengjie.zhang}@ecs.vuw.ac.nz, mark.johnston@msor.vuw.ac.nz

***Abstract*—Order acceptance and scheduling (OAS) is a hard optimisation problem in which both acceptance decisions and scheduling decisions must be considered simultaneously. Designing effective solution methods or heuristics for OAS is not a trivial task, especially to deal with different problem configurations and sizes. This paper proposes a new heuristic framework called forward construction heuristic (FCH) for OAS and develops a new sequential genetic programming (SGPOAS) method for automatic design of FCHs. The key idea of the new GP method is to learn priority rules directly from optimal scheduling decisions at different decision moments and evolve a set of rules for FCHs instead of a single rule as shown in previous studies. The results show that evolved FCHs are significantly better than evolved single priority rules. The evolved FCHs are also competitive with the existing meta-heuristics in the literature and very effective for large problem instances.**

***Index Terms*—genetic programming, scheduling, heuristics**

I. Introduction

Order acceptance and scheduling (OAS) is a planning problem in make-to-order manufacturing systems. OAS aims to determine which customer orders to be accepted and how these orders are scheduled in order to optimise the use of manufacturing resources. As compared to the pure scheduling problems where only scheduling decisions are taken into account, OAS is more complicated because both acceptance and scheduling decisions need to be considered simultaneously to find the optimal solutions. This paper focuses on the OAS problem in the single machine environment with dependent setup times [1], [2], [3]. Given a set of N customer orders, each order $j \in \{1, \ldots, N\}$ is characterised by a release time r_j, a processing time p_j, a due date d_j, a weight/penalty w_j, a maximum revenue e_j, and a deadline $\bar{d_j}$. Before an order j is processed, a (dependent) setup time $s_{i,j}$ is required if order j is processed right after order i ($s_{0,j}$ is the setup time of order j if it is the first order to be processed). If the order is completed before the due date d_j, the profit prt_j obtained from order j is the maximum revenue e_j. Otherwise, prt_j is the remaining profit after deducting the penalty caused by the tardiness $T_j = \max(0, C_j - d_j)$ from e_j, where C_j is the completion time of order j. Generally, the profit obtained by an order j can be calculated by $prt_j = e_j - w_j T_j$. If orders are finished after their deadlines $\bar{d_j}$, no profit is gained and these orders are rejected. The objective of this problem is to maximise the total profit TPR $= \sum_{j \in \mathbb{A}} prt_j$ where $\mathbb{A}$ is the set of accepted orders. A Mixed Integer Linear Programming (MILP) model of this problem is presented in Oguz et al. [4].

Ghosh [5] proved that OAS is NP-hard and previous works have shown that finding optimal solutions in this case is very challenging, even for small instances [1], [6], [7]. Therefore, several heuristics have been proposed to search for near optimal solutions for OAS. Rom and Slotnick [8] developed a hybrid method in which solutions found by a genetic algorithm (GA) are further improved by a local search heuristic. In their proposed GA method, a solution is an array of real numbers corresponding to all orders, which decides the sequence in which orders are processed (orders that do not increase total profit will be rejected). Huang et al. [9] considered the OAS problem with setup costs as a major factor (due dates and penalty costs are ignored) and proposed an algorithm for this problem by formulating it as resource-constrained profitable tour problem. Oguz et al. [4] developed a simulated annealing method (ISFAN) for the OAS problem with dependent setup times in customised packing material producers and showed that their proposed method can find good solutions for large scale problem instances. Cesaret et al. [1] developed a tabu search (TS) method to handle the same OAS problem and the experimental results showed that the proposed TS method outperformed ISFAN in most instances. Lin and Ying [3] developed a hybrid artificial bee colony (ABC) method for OAS, where an effective iterated greedy local search heuristic is employed to enhance the quality of solutions found by ABC. Genetic Programming (GP) [10] has been successfully used for many scheduling problems [11], [12], [13] and it has been also applied in OAS to find reusable and effective scheduling rules to generate initial solutions for search heuristics [2]. The experimental results show that the proposed GP method can improve the effectiveness and efficiency of the search heuristics. Different representations and evaluation schemes in GP for OAS were latter investigated by Park et al. [14]. A comprehensive review of OAS can be found in Slotnick [15].

Previous studies have shown that GP is an effective method to design heuristics for OAS [14] or enhance the performance of other meta-heuristics [2]. However, there are two main limitations with the current GP methods. Because heuristics evolved by GP are restricted to a single priority rule, it

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

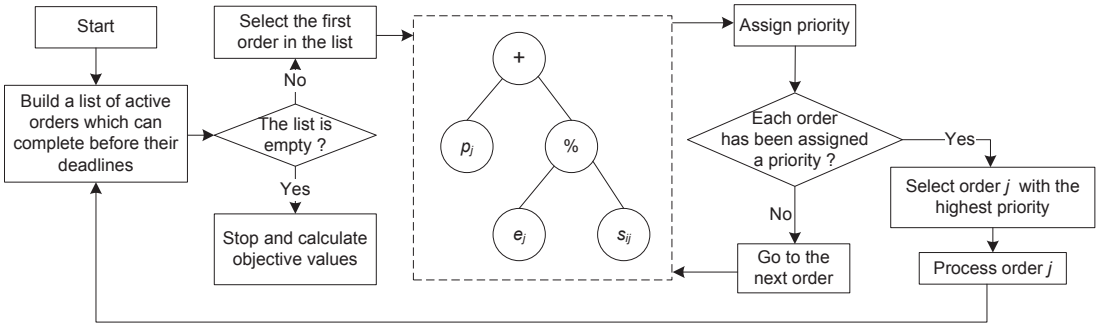


Figure 1: Constructing an active scheduling with the priority rule $p_j + e_j \% s_{i,j}$.

is very difficult for these evolved heuristics to cover all situations of the OAS problem. This is the main reason why the evolved heuristics are usually not as competitive as the highly customised meta-heuristics for OAS. The other limitation is that the fitness of evolved heuristics is measured by their performance on training instances. This approach is quite simple and is able to reflect the average performance of evolved heuristics, at least at the instance level. However, this approach does not point out whether a decision made at a decision moment (at the beginning of the schedule or when an order is completed) is correct or not. As a result, we do not know whether evolved heuristics perform well or badly in a certain situation. Moreover, because the fitness is based on the relative error between the obtained objective values and the referenced objective values (normally an upper bound from solving the MILP model [4]), certain biases may occur (e.g. from loose upper bounds for certain instances). This may make GP evolved heuristics more specialised on certain cases and deteriorate their generality.

A. Goals

The goal of this paper is to develop a new genetic programming method to learn heuristics for OAS based on optimal scheduling data. The novelty of the proposed method is two-fold. First, the fitness of evolved priority rules depends on how well the rules perform at each decision moment rather than the final objective values obtained from each training instance usually used in previous studies. Second, a sequential learning method is developed to learn a set of priority rules to cope with different situations. In order to achieve our goal, the following research objectives need to be fulfilled.

i. Designing forward construction heuristics to build complete OAS solutions based on a set of priority rules.
ii. Generating a dataset of optimal scheduling decisions.
iii. Developing a new genetic programming method to learn a set of priority rules from the generated dataset.

B. Organisation

The rest of this paper is organised as follows. The next section proposes new forward construction heuristics (FCHs) for OAS. Section III describes how the dataset of optimal scheduling decisions is generated using branch-and-bound (B&B). In Section IV, we describe the new GP method [14] for generating a set of rules employed in FCHs. The performance of the proposed GP method is compared to that of the existing GP method in Section V. Then, we compared the performance of evolved FCHs with other meta-heuristics in the literature. Finally, we provide conclusions and discussions for future research in Section VI.

II. Forward Construction Heuristics

As discussed in the previous section, the heuristics evolved by the existing GP method [14] are mainly based on a single priority rule; and therefore, it is very difficult for these heuristics to capture all situations in OAS. In this section, we develop a new forward construction heuristic (FCH) to build a complete OAS solution based on a set of priority rules. The key principle of this heuristic is to combine priority rules specialised for different situations together in order to improve the quality of the final obtained solutions.

A. Constructing active schedule

An example of how a priority rule can generate an OAS solution is shown in Figure 1 (the priority rule is represented in tree-form which is used in GP). The procedure in this figure starts by building a list of unscheduled active orders which can be processed before their deadlines (see the definition of active orders in this paper in the next paragraph). Then, the evolved rule calculates the priority of each order in the list using the corresponding information of that order. After priorities are assigned to all orders in the list, the order with the highest priority will be processed (and certainly this order is accepted). The current time of the schedule (ready time to process the next order) is adjusted. The list of unscheduled active orders is updated and the procedure stops if no order can be completed before its deadline.

Given a set K of unscheduled orders, an order j is called active if $r_j < t(K)$ where $t(K) = \min_{j \in K} \left\{ \max\{r_j, t\} + s_{prev,j} + p_j \right\}$, t is the current decision moment, and $prev$ is the index of the previously completed order. Only active schedules are considered here in order to avoid wasting the available capacity of the machine. It should be noted that active schedules generated in this case do not necessarily contain an optimal schedule because of the dependent setup

Algorithm 1 Pseudocode for FCH

1: let $\mathcal{L}$ (initially empty) be the list of determined rules for each decision moment
2: let S^* be the best found solution
3: **repeat**
4: $\quad\mathcal{L}' \leftarrow \mathcal{L}$
5: $\quad$let S be the partial solution
6: $\quad$**while** $\mathcal{L}'$ is not empty **do**
7: $\quad\quad$select the order j to process next using one loop of the procedure in Figure 1 with the first rule in $\mathcal{L}'$
8: $\quad\quad$insert j into S
9: $\quad\quad$remove the first rule from $\mathcal{L}'$
10: $\quad$**end while**
11: $\quad$**if** there are no unscheduled orders **then**
12: $\quad\quad$break;
13: $\quad$**end if**
14: $\quad$**for all** $\Delta_k^A \in \mathcal{A}$ **do**
15: $\quad\quad S_k \leftarrow S$
16: $\quad\quad$select the order j to process next using one loop of procedure in Figure 1 with Δ_k^A; insert j into S_k
17: $\quad\quad S_k \leftarrow$ applying the procedure in Figure 1 to S_k with Δ_0^A until there is no unscheduled orders
18: $\quad$**end for**
19: $\quad$let $k^* = \text{argmin}_k(total\_profit(S_k))$
20: $\quad$append $\Delta_{k^*}^A$ to $\mathcal{L}$; and $S^* \leftarrow S_{k^*}$
21: **until** termination condition is reached
22: **return** S^*

times. However, it can help us find very good (near optimal) solutions. From our experiments, evolved rules that focus on active orders are more effective than those considering all unscheduled orders [2].

B. FCH procedure

Given a backbone priority rule Δ_0^A and a set of alternative priority rules $\mathcal{A} = \{\Delta_0^A, \Delta_1^A, \Delta_2^A, \ldots, \Delta_M^A\}$, an OAS solution can be constructed using the procedure in Algorithm 1.

The list $\mathcal{L}$ contains priority rules in the sequence in which they are applied to determine the next order to process. For example, if $\mathcal{L} = (\Delta_1^A, \Delta_0^A, \Delta_4^A)$, the first order to process will be determined by apply rule Δ_1^A to calculate priorities for all unscheduled orders and the one with the highest priority will be processed next (as shown in Figure 1). The next two orders to be processed will be determined by Δ_0^A and Δ_4^A. In Algorithm 1, steps 4–10 are used to determine the partial solution with $\mathcal{L}$. If there is no unscheduled or non-profitable orders after all rules in $\mathcal{L}$ are applied, the algorithm is terminated. Otherwise, we try to find the most suitable rule to be applied at the decision moment t at the end of partial solution S. To estimate the effectiveness of rule Δ_k^A at this decision moment, we use the backbone rule Δ_0^A to determine orders in the latter decision moments. The rule $\Delta_{k^*}^A$ which leads to the solution S_{k^*} with the highest total profit will be appended to $\mathcal{L}$.

The procedure in Algorithm 1 is a forward heuristic because we only examine what priority rules should be applied at the latter decision moments without going backward to check whether rules in the previous decision moments should be changed. In order to make FCHs effective, we need a good backbone rule Δ_0^A as well as the alternative set $\mathcal{A}$. A good backbone rule should provide generally good solutions for OAS so we can make good estimates of the effectiveness of rules at a decision moment. Meanwhile, the set of alternative priority rules should cover a diverse set of rules to handle different situations. In Section IV, a new sequential genetic programming method is proposed to help automatically design priority rules used in FCHs.

III. Generating the dataset

Since rules designed for FCHs focus on each decision moment, using the original OAS dataset and the fitness function in previous studies [2], [14] is not suitable because they mainly emphasise on the overall performance of rules for each instance (rather than a particular decision moment). In order to learn which priority rules are more suitable at a particular decision moment, one straightforward approach is to use the scheduling data from optimal OAS solutions. In this paper, a branch-and-bound (B&B) method is developed to generate a dataset of optimal scheduling decisions for OAS.

Branch-and-bound is a popular optimisation approach in scheduling [16]. Basically, there are two important steps in B&B: (1) branching and (2) bounding. The proposed B&B algorithm implicitly enumerates all candidate schedules where the branching strategy and the bounding approach are used to prune/eliminate non-optimal candidate schedules. The basic B&B algorithm proposed in this study can be described as follows:

i. Initialise B&B with a root node containing an empty schedule; the node is marked as unexamined. Set lower bound to $-\infty$.
ii. Examine the bottom-left-most unexamined node of the search tree.
iii. Calculate an upper bound (the highest total profit that can be obtained) for the node with its current partial schedule and unscheduled orders.
iv. If the upper bound is lower than the lower bound, the node will be pruned (no longer explored).
v. If the upper bound is equal to the lower bound or there is no unscheduled order, the partial schedule from the node is a complete schedule and the lower bound is updated if the total profit obtained from the considered schedule is higher than the lower bound.
vi. If the upper bound is higher than the lower bound, apply the branching strategy to generate child nodes by appending unscheduled orders to the partial schedule of the parent node.
vii. Return to step ii until there is no unexamined node.

A. Upper bound

In their study, Oguz et al. [4] have proposed two upper bounds for OAS to evaluate the performance of their proposed

heuristics [4], [1]. The first bound is generated by using their MILP model with the time limit of 3600 seconds while the second bound is obtained by solving the linear programming (LP) relaxation of the MILP strengthened with some additional valid inequalities. Although these bounds are effective in some cases, they are too computationally expensive to use within B&B. Therefore, we proposed a new simple and efficient approach to determining an upper bound for OAS. This approach is motivated by the observation that the OAS problem with a single machine is similar to the conventional knapsack problem where a set of items are to be selected to maximise the total value while the total weight does not exceed a given limit. Given a set K of unscheduled orders at the decision moment t, a MILP of a simplified OAS model is as follows:

$$\text{maximise} \sum_{j \in L} e'_j I_j \tag{1}$$

subject to:

$$\sum_{j \in L} p'_j I_j \leq T \tag{2}$$

$$I_j \in \{0,1\} \quad \forall j \in L \tag{3}$$

where I_j is the decision variables, which determine whether each order j is accepted ($I_j = 1$) or rejected ($I_j = 0$). All orders $j \in K$ with $C'_j < \bar{d}_j$ are included in L where $C'_j = \max\{r_j, t\} + \min_{i \in K'}\{s_{i,j}\} + p_j$ is the earliest possible completion time of order j and $K' = K \cup \{prev\}$ and $prev$ is the index of the previous order that has just been finished. The modified profit $e'_j = e_j - w_j \max\{0, C'_j - d_j\}$ is the highest profit that can be obtained by order j. The modified processing time $p'_j = \min_{i \in K'}\{s_{i,j}\} + p_j$ is the least possible time to process order j. Finally, $T = \max_{i \in K}\{\bar{d}_i\} - \min\{t, \min_{i \in K}\{r_i\}\}$ is the total time budget to process all orders.

In this MILP model, only acceptance/rejection decisions are considered and scheduling decisions are ignored. This simplified model is in the form of a knapsack problem which can be solved efficiently by some off-the-shelf optimisation approaches. In this study, dynamic programming [17] is used to solve the knapsack problem because it is effective in a wide range of problem instances. Other more sophisticated and efficient optimisation approaches such as branch-and-bound can also be used here to solve the knapsack problem. In our B&B algorithm, the upper bound is the optimal objective value obtained from the knapsack problem plus the total profit obtained from the scheduled orders in the partial solution. From our experiments, the upper bound obtained from our knapsack model is usually very close to those obtained from the LP relaxation and the MILP model of OAS [4], [1], and its computational times are much lower.

B. Branching strategy

In order to make B&B more efficient, we need to design a good branching strategy. We apply these two rules:

i. All orders that cannot be completed before their deadline will not be considered ($C'_j < \bar{d}_j$).

ii. Only considering orders that satisfy $r_j < EC$ or $\max_{i \in K}\{s_{i,j}\} > s_{prev,j}$, where EC is the earliest completion time of all remaining orders $j \in K$.

The first rule is quite straightforward because we do not want to consider orders that cannot provide any profit. Meanwhile, the second rule tries to eliminate orders that would be too early to process now. The first condition in this rule is similar to that used to generate active schedules [16]. However, because of the dependent setup times, the first condition becomes too strict and we need to further check if the considered order can have lower setup times if it is processed later. Therefore, we only ignore a node if the order appended to its partial solution fails to satisfy these two conditions.

Moreover, child nodes generated from a parent node are sorted based on the priority rule Δ_B in equation (4) such that nodes with higher priorities will be explored first. This rule is applied to quickly obtain a very strong lower bound that can help to eliminate many unexamined nodes (via pruning).

$$\Delta_B = \frac{e_j / \bar{d}_j}{h \times p_j - s_{prev,j} \min(t - d_j, 0)} \tag{4}$$

The behaviour of this rule can be summarised as follows.

- If two orders have equivalent values of slack $(t - d_j)$ and processing/setup times, the order with a higher revenue and closer deadline will be processed first.
- If two orders have similar revenue and deadline, setup time is ignored when they are late ($t > d_j$). If they are early, the one with the smaller setup time and slack will have higher priority.

The coefficient h is used to control the importance of processing time as compared to setup times and slacks. From our experiments, $h = 7.5$ provides reasonable results across different OAS instances.

After an optimal solution of an instance is obtained by the proposed B&B algorithm, we will know which order needs to be processed at a decision moment. The dataset of optimal scheduling decisions is presented as shown in Figure 2. At the decision moment $t_n = 23$ in Figure 2 with the optimal sequence of remaining orders $(5, 4, 3, 10, 7, 9, 2, 1)$, order 5 must be the one with the highest priority at this decision moment and it must be processed before other remaining orders. If it is viewed as a classification problem, seven classification decisions need to be made, corresponding to seven pairwise comparison $\langle a, b \rangle$ where a and b are the indices of two orders. In this paper, we arrange the data such that $a < b$ and the seven pairwise comparisons at the decision moment t_n as shown in Figure 2 are $\langle 4, 5 \rangle$, $\langle 3, 5 \rangle$, $\langle 5, 10 \rangle$, $\langle 5, 7 \rangle$ $\langle 5, 9 \rangle$, $\langle 2, 5 \rangle$, and $\langle 1, 5 \rangle$. The last column shows the classification decision which is 1 if order a is processed before order b and 0 otherwise.

In this paper, the proposed B&B method is used to solve all (250) instances with 10 orders from Cesaret et al. [1] and the optimal scheduling data obtained from these instances will be

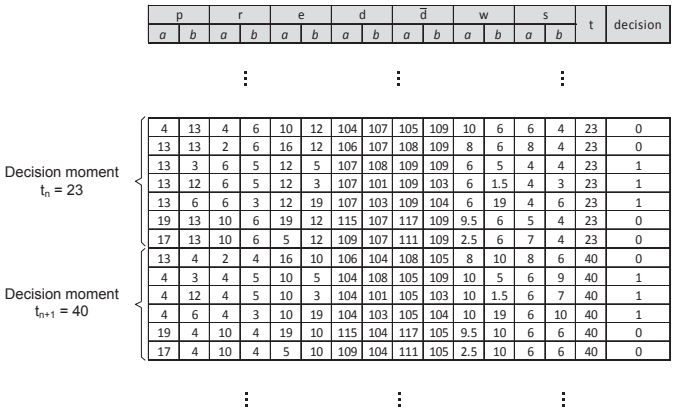


Figure 2: Dataset of optimal scheduling decisions.

used by our proposed GP method in the next section to learn a set of rules for FCHs.

IV. Sequential Genetic Programming Method

In this section, we describe the GP method for learning priority rules through the training set of optimal scheduling decisions.

A. Representation

The proposed GP method uses the same representation as in previous studies [2], [14] to evolve priority rules for OAS. In this case, the priority rule is in tree form as shown in Figure 1. The terminal set and function set used to synthesise priority rules are presented in Table I. In this table, the protected division function $\%$ returns a value of 1 when division by 0 is attempted.

For each training example (a row in Figure 2), the priority rule Δ will be used to the assign priority $\Delta(a)$ for order a and the assign priority $\Delta(b)$ for order b. For a specific training example, if $\Delta(a) > \Delta(b)$ ($\Delta(a) < \Delta(b)$) when the decision is 1 (0), the priority rule makes correct decision for that example ($\Delta(a) = \Delta(b)$ means the priority rule makes incorrect decision because it cannot discriminate between the two orders).

B. Fitness function

Different from previous studies where fitness of an evolved priority rule is measured by the average relative error between the objective values and upper bound of training instances, the fitness in our proposed GP method is measured by the accuracy of the decision made by evolved rules in each training example. After an evolved priority rule Δ is applied to all training instances, the fitness is calculated as follows:

$$fitness(\Delta) = \alpha Acc_1 + (1-\alpha)Acc_2 \tag{5}$$

where

$$Acc_1 = \frac{\text{number of correct moments}}{\text{total number of decision moments}} \tag{6}$$

Table I: Terminal and function sets for priority rules

Symbol	Description	Symbol	Description
R	release time r_j	P	processing time p_j
E	revenue e_j	W	penalty w_j
S	setup time $s_{prev,j}$	d	due date d_j
D	deadline $\bar{d_j}$	t	current time
#	random number from 0 to 1		
Function set	$+,-,\times,\%$ (protected division)		

and

$$Acc_2 = \frac{\text{number of correct examples}}{\text{total number of examples}} \tag{7}$$

In our fitness function, Acc_1 tries to measure the accuracy of priority rule for each decision moment. In this case, a correct moment is the decision moment in which the priority rule provides the correct decisions for all examples. Meanwhile, Acc_2 aims to measure the accuracy of each pairwise comparison. While Acc_1 emphasises more on identifying the order with the highest priority at a decision moment, Acc_2 emphasises more on eliminating orders that are unlikely to be processed at a decision moment. In general, if a priority rule determines the optimal solutions for the OAS problem, the fitness for that priority rule is 1 ($Acc_1 = Acc_2 = 1$). Otherwise, rules with higher fitness are better because they can produce solutions more similar to the optimal solutions. In this paper, Acc_1 is more important as we want evolved rules to make correct decision at each decision moment. Acc_2 is used in the fitness in order to help smoothen the fitness function to avoid GP from being stuck at a large plateau.

Although the fitness function proposed here is very different from the one used in previous studies, they both aim to help GP evolve priority rules that give the highest priority to the order which should be processed next. This property of the evolved rules is still maintained by the new fitness function because orders with higher priorities are preferred in pairwise comparisons (as shown in Section IV-A).

C. Proposed algorithm

The overall algorithm of the proposed sequential genetic programming method for OAS (SGPOAS) is presented in Algorithm 2. The algorithm starts by loading all training examples in the dataset $\mathcal{D}$ (see Section III). The list $\mathcal{A}$ of alternative rules for FCHs is initialised with only the rule Δ_B in equation (4). In each loop of SGPOAS (steps 4–19), a GP process is performed to find the rule with the best fitness based on the training set $\mathcal{D}$. The GP process in SGPOAS is similar to the conventional tree-based GP (see Section IV-D for more information about the parameters and genetic operators of GP). At the end of each GP process, the decision moments in which the best evolved rule Δ^* makes correct decisions are identified and all training examples belonging to these moments are removed from the training set $\mathcal{D}$. SGPOAS continues until the training set $\mathcal{D}$ becomes empty.

The sequential approach is applied here because it is very hard to find a rule which is capable of providing optimal

Algorithm 2 Sequential Genetic Programming (SGPOAS)

let $\mathcal{D} \leftarrow \{E_1, E_2, \ldots, E_T\}$ be the training data
$\mathcal{A} \leftarrow \{\Delta_0^A\}$ where $\Delta_0^A = \Delta_B$ (see equation (4))
repeat
  //a new GP process
  randomly initialise the population $P \leftarrow \{\Delta_1, \ldots, \Delta_S\}$
  set $\Delta^* \leftarrow null$ and $fitness(\Delta^*) = -\infty$
  while $generation \leq maxGenerations$ **do**
    for all $\Delta_i \in P$ **do**
      apply Δ_i to all examples $E_t \in \mathcal{D}$
      evaluate $fitness(\Delta_i)$
      if $fitness(\Delta_i) > fitness(\Delta^*)$ **then**
        $\Delta^* \leftarrow \Delta_i$
        $fitness(\Delta^*) \leftarrow fitness(\Delta_i)$
      end if
    end for
    $P \leftarrow$ apply reproduction, crossover, mutation to P
    $generation \leftarrow generation + 1$
  end while
  //update the training set
  $\mathcal{D} \leftarrow$ remove all examples belonging to correct moments, determined by Δ^*, from $\mathcal{D}$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{\Delta^*\}$
until $\mathcal{D}$ is empty
return $\mathcal{A}$

decisions for all situations. In each loop of SGPOAS, we try to find a rule that can provide optimal scheduling decisions for most training examples. Because a different rule (specialised for certain situations) is evolved in each loop of SGPOAS, the list $\mathcal{A}$ will contain a set of rules that can cope with different situations and they can also compensate each other.

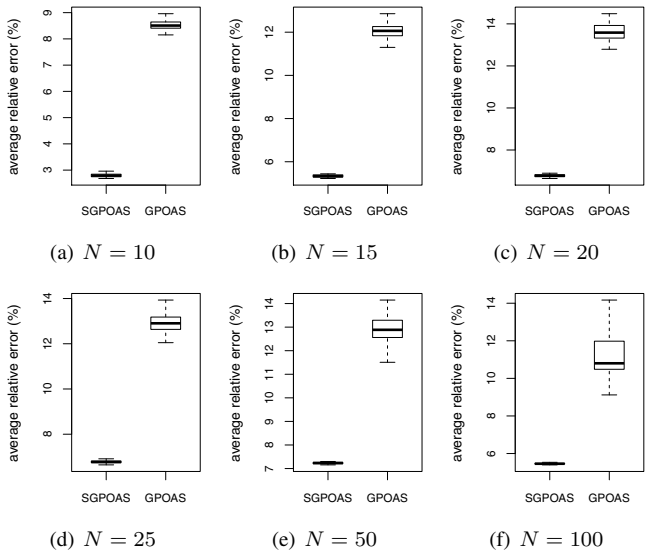
Since the rule Δ_B is used as the branching strategy in B&B for generating training data, it is expected that the optimal solution found by B&B will be influenced by this rule. Therefore, it is helpful to use Δ_B as the backbone rule for FCHs. Basically, we want SGPOAS to learn how B&B searches for optimal solutions and to embed this knowledge into the evolved FCHs. Similar to B&B, evolved FCHs try to correct what went wrong with the solutions generated by Δ_B.

D. Parameter Settings

To generate the initial GP population, we applied the *ramped-half-and-half* method [10]. Subtree crossover and subtree mutation [10] are employed in this study. The crossover rate, mutation rate and reproduction rate used in GP are 90%, 5% and 5%, respectively. The maximum depth of GP trees is 8. Tournament selection of size 5 is used to select individuals for genetic operators. The population size of 1000 and $maxGenerations$ of 50 are used for each GP process. For the fitness function of SGPOAS, we set $\alpha = 0.9$ in order to emphasise more on the importance of making correct decisions at a moment.

V. Computational Results

In order to evaluate the performance of SGPOAS, it is compared to an existing GP method for OAS (GPOAS) [14]. GPOAS also uses the whole dataset from Casaret et al. [1] with 10 orders for training priority rules. Each GP method will perform 50 independent runs and the datasets from Casaret et al. [1] (with number of orders N from 25 to 100) are used to assess the performance of evolved FCHs from SGPOAS and evolved priority rules from GPOAS. This section also compares the performance of evolved FCHs against those of simulated annealing (ISFAN) [4] and tabu search (TS) [1]. All statistical tests discussed in this study are Wilcoxon tests and they are considered significant if the obtained p-value is less than 0.05.



(a) $N = 10$ (b) $N = 15$ (c) $N = 20$

(d) $N = 25$ (e) $N = 50$ (f) $N = 100$

Figure 3: Performance of SGPOAS and GPOAS.

A. SGPOAS and GPOAS

For each dataset with a fixed number of orders N, an evolved FCH or an evolved priority rule is applied to 250 instances (classified into 25 subsets based on 5 different tardiness factors τ and due date ranges R [1]). Due to the space limitation of this paper, we will not present all detailed results for each subset. Figure 3 shows the average absolute percentage error of evolved FCHs and priority rules for all 250 instances in the dataset with different number of orders N. The absolute percentage error is $100 \times (UB - TPR)/UB$ where UB is the upper bound found by CPLEX [1], [4] and TPR is the total profit of the solution found by FCHs or priority rules. It is noted that the GP method with lower average absolute percentage errors is better.

From the results in Figure 3, it is easy to see that FCHs evolved by SGPOAS are significantly better than priority rules evolved by GPOAS. There is an interesting point from these results. The gaps between SGPOAS and GPOAS are smaller and the variance of results from GPOAS tend to increase when the number of orders increases. One reason is that the acceptance decision is critical when the number of orders is low and multiple rules in FCHs can help handle this issue better than single priority rule evolved by GPOAS. When the number of orders is high, it is very difficult for a single rule to cope with all situations and provide good results.

This is when FCHs show their advantages over single priority rules. The detailed results (not presented here) for each subset (with different τ and R) also show that evolved FCHs are significantly better than evolved single priority rules.

Regarding the computational time, it takes 178 seconds for SGPOAS and 88 seconds for GPOAS on average to complete one training run. However, because SGPOAS and GPOAS can learn rules or set of rules offline, the difference in running times between the two GP methods are not critical. Similarly, the running times of FCHs are longer than those of single priority rules but they are negligible because these rules are able to solve large instances (up to 100 orders) within one second. Given that evolved FCHs are much better than single priority rules, extra times used for SGPOAS to learn a set of rules are reasonable and worthwhile.

B. Evolved FCHs and other meta-heuristics

Figure 4 shows the performance of TS, ISFAN and evolved FCHs across all instances in the datasets with different number of orders. In this figure, the FCH line represents the average performance of all evolved FCHs and the shaded area surrounding this line represents the minimum and maximum average relative error (%) of evolved FCHs. In general, the performance of evolved FCHs from different independent runs

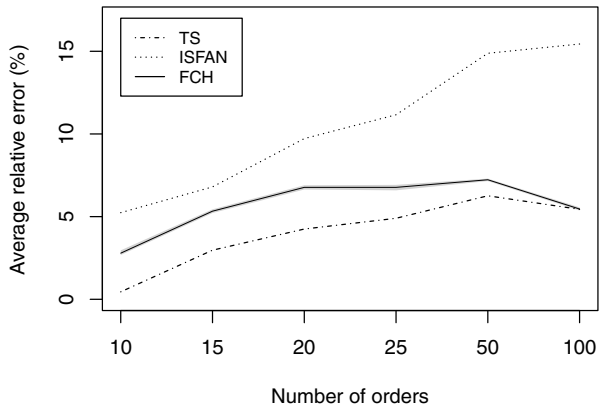


Figure 4: Performance of evolved FCHs, TS and ISFAN.

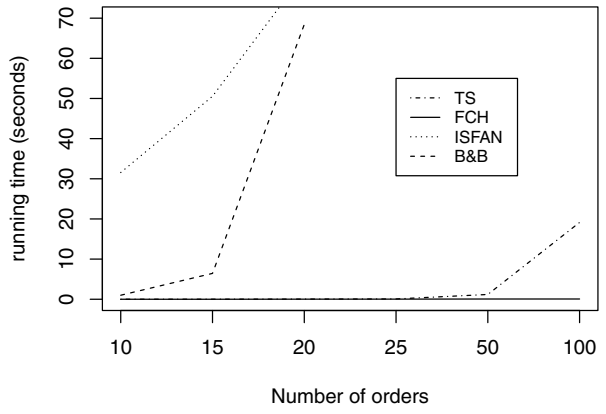


Figure 5: Running times of evolved FCHs, TS and ISFAN.

Table II: % Deviations from upper bound of total profit obtained by ISFAN, TS and FCH ($N = 100$)

τ	R	ISFAN			TS			FCH*		
		min	avg	max	min	avg	max	min	avg	max
0.1	0.1	8	9	13	1	2	3	1	2	2
	0.3	7	9	10	2	2	3	1	1	2
	0.5	6	9	11	0	1	3	0	1	1
	0.7	6	9	12	0	0	1	0	0	0
	0.9	8	12	16	0	0	1	0	0	0
0.3	0.1	10	12	13	1	3	4	1	2	4
	0.3	11	13	17	2	3	5	1	3	4
	0.5	10	14	17	1	2	4	1	2	3
	0.7	12	14	15	0	2	3	0	1	2
	0.9	9	13	17	0	1	2	0	1	1
0.5	0.1	12	16	18	2	4	5	3	4	5
	0.3	12	15	18	3	4	6	3	4	6
	0.5	14	17	19	3	4	5	3	4	6
	0.7	13	17	21	2	3	4	1	3	4
	0.9	12	18	24	1	2	5	1	2	4
0.7	0.1	13	17	19	3	5	6	4	5	7
	0.3	13	17	21	4	7	11	3	7	11
	0.5	14	18	24	4	6	13	4	7	17
	0.7	16	19	23	3	7	13	4	8	11
	0.9	15	18	24	5	8	13	4	8	12
0.9	0.1	14	17	20	7	9	12	7	10	13
	0.3	16	20	26	7	14	17	10	13	16
	0.5	19	21	25	11	16	18	11	16	20
	0.7	15	21	24	11	15	20	11	16	19
	0.9	13	21	28	11	16	22	6	15	22

of SGPOAS are quite consistent as the performance range is very small. From the figure, it is obvious that evolved FCHs are always better than ISFAN on all problem sizes and the gaps between the performances of ISFAN and FCHs increase as the number of orders increases. The performance of evolved FCHs is about 2–3% worse than that of TS when the number of orders is lower than 50. However, it is noted that the gaps between TS and evolved FCHs gradually reduce as the number of order increases. When the number of orders is 100, TS is dominated by a number of evolved FCHs. Table II illustrates the performances one of the best evolved FCHs (based on its performance on the entire dataset) as compared to ISFAN and TS on the dataset with $N = 100$. We can see that this evolved FCH totally dominates ISFAN on all problem configurations. It is also very competitive with TS on instances with high τ and better than TS on instances with low τ.

The computational times of ISFAN, TS, FCHs and the proposed B&B (in Section III) are presented in Figure 5. ISFAN is obviously the least efficient method here for instances even with a small number of orders. With the number of orders of 10 to 20, the running times of ISFAN is even worse than those of B&B. The computational times of TS and evolved FCHs are very small for instances with less than 50 orders. The difference in running times between TS and FCHs is only noticeable when the number of orders are from 50 to 100. It is clear that evolved FCHs are still very efficient for large instances. For the instances with 100 orders, it takes TS about 19 seconds on average to solve an instance while FCHs only spends 0.064 seconds.

C. Further discussion

This paper provides an good opportunity to compare three different types of approaches for a hard computational problem, which are exact optimisation (B&B), meta-heuristics

(i.e. TS; we ignore ISFAN in this discussion because of its poor performance), and automatic heuristic design (SGPOAS). Figures 4–5 are very helpful to help us understand advantages and disadvantages of the three approaches.

For small instances (number of orders $N \leq 15$), B&B is a suitable method because it can provide the optimal solutions within reasonable times. When $N = 20$, B&B failed to obtain the optimal solutions within a five minute time limit for some instances but its feasible solutions found (lower bound) are still better than TS. However, when $N > 20$, the computational time of B&B increases rapidly. With N from 25 to 50, TS is a good solution method as it balances between the computational time and the quality of found solutions. For large instances with $N = 100$, TS becomes cumbersome and evolved FCHs become a good choice because they are both effective and efficient in these cases.

This observation suggests that choosing an approach to dealing with hard computational problems is very critical. From our experiments, it is clear that problem size is an important factor that influences the choice of a suitable approach. If the practical problem is reasonably small and exact methods are capable of solving it within the time limit, exact methods are always the excellent choice as they can guarantee optimal solutions or at least show us the gaps between the obtained solutions and the upper (lower) bound. When the running time of exact methods are out of control, off-the-shelf meta-heuristics such as tabu search and evolutionary algorithms are needed in order to find acceptable solutions. However, if the problem is large and even meta-heuristics are not effective enough, it is a good idea to use an automatic heuristic design approach such as SGPOAS in this paper to extract the knowledge of optimal or good solutions to design efficient heuristics. These heuristics can either operate on their own or can be used to enhance the effectiveness and efficiency of meta-heuristics.

VI. Conclusions

This study develops a new sequential genetic programming method that evolves forward construction heuristics for OAS. They key difference between the proposed SGPOAS in this paper and the existing GP method is that the priority rules are trained based on optimal scheduling decisions rather than the objective values obtained from training instances. Moreover, the proposed GP method aims to evolve a set of rules instead of a single priority rule. The experimental results show that FCHs evolved by SGPOAS are significantly better than evolved single rules in all testing datasets. The evolved FCHs are also competitive with existing meta-heuristics. The results also show that evolved FCHs are a very suitable approach to dealing with large problem instances of OAS.

There are many aspects that need to be investigated in the future studies. First, the backbone rules can also be co-evolved with the set of alternative rules in FCHs in order to enhance the effectiveness of evolved FCHs. Second, it is useful to investigate a new approach for learning a set of alternative rules instead of the sequential learning approach in SGPOAS. The goal should be to reduce the computational time of GP, reduce the number of rules and improve the performance of evolved FCHs. Finally, the generated dataset has basically converted OAS into a binary classification problem in which the classification decision is which order within two specific orders must be processed first. If we are able to achieve 100% classification accuracy, we can use the trained classifier to construct optimal solutions for OAS. Therefore, it is interesting to tackle the heuristic design problem from the classification viewpoint and employ state-of-the-art classification techniques to help us improve the quality of the generated heuristics.

References

[1] B. Cesaret, C. Oguz, and F. S. Salman, "A tabu search algorithm for order acceptance and scheduling," *Computers & Operations Research*, vol. 39, no. 6, pp. 1197–1205, 2012.

[2] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Learning reusable initial solutions for multi-objective order acceptance and scheduling problems with genetic programming," in *Proceedings of the 16th European Conference on Genetic Programming*, 2013, pp. 157–168.

[3] S. W. Lin and K. C. Ying, "Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm," *Journal of the Operational Research Society*, vol. 64, pp. 293–311, 2013.

[4] C. Oguz, F. Sibel Salman, and Z. Bilginturk Yalcin, "Order acceptance and scheduling decisions in make-to-order systems," *International Journal of Production Economics*, vol. 125, no. 1, pp. 200–211, 2010.

[5] J. B. Ghosh, "Job selection in a heavily loaded shop," *Computers & Operations Research*, vol. 24, no. 2, pp. 141–145, 1997.

[6] S. A. Slotnick and T. E. Morton, "Selecting jobs for a heavily loaded shop with lateness penalties," *Computers & Operations Research*, vol. 23, no. 2, pp. 131–140, 1996.

[7] ——, "Order acceptance with weighted tardiness," *Computers & Operations Research*, vol. 34, no. 10, pp. 3029–3042, 2007.

[8] W. O. Rom and S. A. Slotnick, "Order acceptance using genetic algorithms," *Computers & Operations Research*, vol. 36, no. 6, pp. 1758–1767, 2009.

[9] S. Huang, M. Lu, and G. Wan, "Integrated order selection and production scheduling under mto strategy," *International Journal of Production Research*, vol. 49, no. 13, pp. 4085–4101, 2011.

[10] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[11] C. D. Geiger, R. Uzsoy, and H. Aytug, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Heuristics*, vol. 9, no. 1, pp. 7–34, 2006.

[12] D. Jakobovic, L. Jelenkovic, and L. Budin, "Genetic programming heuristics for multiple machine scheduling," in *EuroGP'07: Proceedings of the 10th European Conference on Genetic programming*, 2007, pp. 321–330.

[13] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter, "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems," *International Journal of Production Economics*, 2012, in Press. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925527312004574

[14] J. Park, S. Nguyen, M. Zhang, and M. Johnston, "Genetic programming for order acceptance and scheduling," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2013, pp. 3261–3268.

[15] S. A. Slotnick, "Order acceptance and scheduling: A taxonomy and review," *European Journal of Operational Research*, vol. 212, no. 1, pp. 1–11, 2011.

[16] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.

[17] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York: John Wiley & Sons, Inc., 1990.