Learning Single-Machine Scheduling Heuristics Subject to Machine Breakdowns with Genetic Programming

Wen-Jun Yin, Min Liu, Cheng Wu
Department of Automation, Tsinghua University, Beijing 10084 China
ywj99@mails.tsinghua.edu.cn, liumin@cims.tsinghua.edu.cn, wuc@tsinghua.edu.cn

Abstract- Genetic Programming (GP) has been rarely applied to scheduling problems. In this paper the use of GP to learn single-machine predictive scheduling (PS) heuristics with stochastic breakdowns is investigated, where both tardiness and stability objectives in face of machine failures are considered. The proposed bi-tree structured representation scheme makes it possible to search sequcing and idle time inserting programs integratedly. Empirical results in different uncertain environments show that GP can evolve high quality PS heuristics effectively. The roles of inserted idle time are then analysed with respect to various weighting objectives. Finally some guides are supplied for PS design based on GP-evolved heuristics.

1 Introduction

Production scheduling problems have attracted many research efforts over the last several decades (Conway *et al* 1967, Roadmmer *et al* 1988). However one important factor that prevents many algorithms from being practicable is that they have poor mechanisms to handle uncertainties in manufacturing systems such as machine breakdowns (McKay *et al* 1989).

In real world, there are often two key elements in the scheduling systems considering exceptions, i.e., schedule genereation and revisions. Schedule generation determines the beginning and completing times of all operations of jobs in a given time window and then releases the plan to shop floor. As time passes, the initial schedule might be revised (rescheduled) by the second element when unforseen disruptions such as breakdowns occur. From the viewpoints of planners, those simple and quick heuristics for scheduling and rescheduling will be preferable in practice. Dispatching rules such as SPT (shortest processing time first) and EDD (earliest duedate first) are typical examples. However, the quality of the obtained schedules using these rules is often difficult to be assured, especially in face of disturbances.

On the other hand, the initial schedule is planned not only to make efficient use of shop resources for high performance but also to guide external activities such as material procurement or downstream processes. If the released schedule is modified due to disruptions, those external activities may be affected as well. Hence it is necessary to consider machine failures in the initial schedule so as to retain stability without loss of efficiency meanwhile (Wu *et al* 1993, O'Donovan *et al* 1999). The methodologies based on this idea are sometimes called Predictable Scheduling and the initial schedule is called predictive schedule because events that may or may not happen in the future will be perceived and considered in it. Apparently, this idea is different from common robust scheduling which often focuses on shop performance measures (Daniels and Kouvelis, 1995). In building a predictive schedule (PS), one promising way is to deliberately keep machine idle for appropriate time even when jobs become available. If one breakdown occures and is then repaired just during the idle period, the disruption will be absorbed by the inserted idle time and make little influence on shop performance or planned external activities. Although it seems not effective to insert idle times in predictive schedule from classical research viewpoints, predictive schedules with approriate idle times will in fact arrive at global balances between producing efficiency and stability and be preferred by practitioners. However, to build predictive schedules with high quality, one needs to determine both job processing sequence and inserted idle time before each job in an integrated manner according to current job and machine status. There is not too many relevant reports on PS yet.

In order to solve complex real-time scheduling problems especially under stochastic breakdowns, the heuristics such as dispatching rules well-customized for specific scenarios can be effective and practical methods. Unfortunately it is often too difficult to develop simple efficient heuristics even by domain experts. It may takes long time and huge efforts to achieve good heuristics. In this paper, we use genetic programming (GP) to discover good PS heuristics for single machine with breakdowns. As one member of the evolutionary computation family, GP (Koza 1992, Banzhaf *et al* 1998) often follows the basic flow like genetic algorithms. But the main difference is that GP evolves computer programs of variable size. Thus if the solution of one problem can be represented by a computer program, GP will be able to search for it. In this sense, GP serves as one potential means for machine learning, rule discovery and data mining (Dimopoulos *et al* 2001). Apparently, we can treat

PS heuristics as programs and then use GP to find good ones.

In Section 2, the scheduling problems of single machine to minimize the mean tardiness subject to breakdowns are formulated. The performance measures concerning both shop efficiency and stability are then defined. The GP learning system is then proposed thoroughly in Section 3 with the focuses on bi-tree structured representation, stochastic sampling and fitness evaluation. The test results are analyzed in detail in Section 4 and some further discussions about idle time and evolved rules are also made. Conclusions and future work are pointed out finally.

2 Problem Formulation

2.1 Predictive Scheduling and Rescheduling

Given a problem instance q with N jobs to be processed on a single machine. The release time r_i, processing time p_i and duedate d_i for each job $i \in N$ are deterministic and known *a priori*. What can be known about breakdowns are their probability distributions of occurring frequencies and durations which may be obtained from previous maintenance records and are assumed to be known *a priori* in the following.

Let $S_p(q,h) = \{\pi_1\pi_2\cdots\pi_N\}$ be the predictive schedule of problem q using some heuristic h, where permutation $\{\pi_1\pi_2\cdots\pi_N\}$ is the processing sequence of N jobs. For each job i in the sequence, denote b_{pi} its beginning time, c_{pi} its completing time and it_i the inserted idle time before it. We have

$$\begin{aligned} b_{p\pi_i} &= \max\{c_{p\pi_{i-1}}, r_{\pi_i}\} + it_{\pi_i} \\ c_{p\pi_i} &= b_{p\pi_i} + p_{\pi_i} \qquad (1 \le i \le N) \end{aligned} \tag{1}$$

where $b_{p\pi_0} = c_{p\pi_0} = 0$.

The initial schedule $S_p(q,h)$ is then released to the shop floor and modified(rescheduled) if necessary. When all jobs are completed, we have a realized schedule $S_r(q)$ where the real completing time of each job i is c_{ri}. Since we focus on obtaining effective predictive scheduling heuristics, the rescheduling strategy adopted here is right-shift rescheduling and the preempt-resume case is assumed. That is, the job in process can be resumed without loss of prior work as soon as the machine is repaired and the sequence in the predictive schedule is still maintained.

2.2 Performance Measures

For problem instance q under stochastic breakdowns, the tardiness of a given job i using heuristic h is defined as

$$T_i(q,h) = \max\{\max(c_{pi}, c_{ri}) - d_i, 0\} \tag{2}$$

where both the planned and realized completing time of job i are considerd because they serve as bases for calculating job tardiness at the beginning and end of planning horizon respectively in practice.

The mean tardiness of the problem with h is then defined as

$$\overline{T(q,h)} = \frac{1}{N}\sum_{i=1}^{N} T_i(q,h) \tag{3}$$

To reduce the impact of breakdowns on external activities, the realized completing time of each job is hoped not too different from the orginal plan. Thus, we can measure the stability of heuristic h on problem q by the mean completing time deviations between the intial and final schedules. That is

$$\overline{CD(q,h)} = \frac{1}{N}\sum_{i=1}^{N} |c_{pi} - c_{ri}| \tag{4}$$

For a given instance q, the effect of predictive scheduling with heuristic h on shop performance and stability could be measured by (3) and (4) respectively. These bicriteria can be combined into a single objective weighted by $w_T \in [0,1]$,

$$f(q,h) = w_T\overline{T(q,h)} + (1 - w_T)\overline{CD(q,h)} \tag{5}$$

Furthermore, the performance of h on certain problem instance set $Q = \{q\}$ can be calculated by

$$F_Q(h) = \frac{1}{|Q|}\sum_{q \in Q} f(q,h) \tag{6}$$

where $|Q|$ is the size of set Q. The less $F_Q(h)$ becomes, the better h acts on Q.

3 Genetic Programming for PS Heuristics

In GP, each individual (program) takes a number of inputs that are relevant to the considered problem, manipulates them through a number of functions and then outputs the requried results. For PS heuristics, it is hoped to get both job sequence and inserted idle time before each job from available shop information, which will be achieved in the following through the chosen terminal and function sets and appropriate individual structures.

3.1 Terminals and Functions

The search space in GP is the space of all possible programs composed of terminals and functions. When applying GP to predictive scheduling problems of a single machine with breakdowns, we choose the following scheduling attributes as to form the terminal set.

- p_i - processing time of job i
- d_i - duedate of job i
- r_i - release time of job i

- N - total job number
- N_r - remained job number
- $RPSum$ - sum of processing time of remained jobs
- $RDSum$ - sum of duedate of remained jobs
- BP - expected period of breakdown occurrence
- MP - expected duration of breakdown
- t - current time
- $Const$ - constant integer in [-5,5].

And the function set is comprised of the following operations

- ($+,-,\times,\%$)- addition, subtraction, multiplication and division where $\%$ refers to protected division which will return 1 if the value of the denominator equals 0
- (min, max)- minimum and maximum of two numbers
- δ - $\delta(x,y)=1$ if $x>y$ and 0 otherwise.

3.2 Bi-tree Structured Representation

In GP, each possible PS heuristic is treated as a candidate computer program (individual) to be evolved. The program is represented with two subtrees,

$$prog=\{ptree,itree\} \quad (7)$$

where each subtree serves as a function that collects the input information to calculates the output (See in Fig. 1). For one problem instance q, subtree $ptree$ calculates the priority index $pri(i,t)$ of each available job i on current time t and subtree $itree$ then output the to-be-inserted idle time $itime(i^*,t)$ before processing i^*, the job of the highest priority index. Obviously,

$$\begin{aligned} b_{pi^*} &= t+itime(i^*,t) \\ c_{pi^*} &= b_{pi^*}+p_{i^*} \end{aligned} \quad (8)$$

Update the simulating time as $t=c_{pi^*}$ and repeat selecting the remaining jobs according to the above procedure untill all jobs have been planned. Then one predictive schedule $S_p(q,prog)$ is obtained.

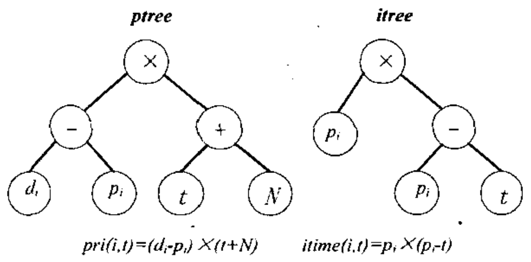
ptree
itree
×
-
+
d_i
p_i
t
N
×
p_i
-
p_i
t
$pri(i,t)=(d_i-p_i)\times(t+N)$
$itime(i,t)=p_i\times(p_i-t)$

Fig. 1 Demonstration of bi-tree structured representation

What can be seen is that the bi-tree structured representation scheme integrates two subprocesses of PS by two subtrees in a more nature way. That is, each time one job is selected and the inserted idle time is then calculated. These two subtrees co-evolve in the space of all possible PS programs with their size and shape changed dynamically. With the help of GP searching mechanisms like genetic algorithms, optimal or near-optimal PS heuristics will be achieved.

3.3 Stochastic Sampling and Fitness Evaluation

The fitness of individual (program) is determined by its test quality on certain set of fitness cases $Q=\{q\}$ according to Formula (6).

However, when the size of learning problem set Q is too large, extensive computational resources will be spent on calculting fitness. To save time-consumption, we modified the fitness evaluation procedure with stochastic sampling (SS) techniques. During the process of SS evaluation on a program, one random subset of Q is firstly generated. The fitness value is then acquired from (6) based on this smaller subset. Another consideration of using SS here is to avoid over-fitting phenomena during learning.

3.4 Configurations of GP

The proposed GP for learning PS heuristics is configed as follows.

Homologous Subtree Crossover. Crossover is carried out between the subtrees of similar functions, i.e., one parent $ptree$ recombines with the other $ptree$ and so does $itree$. The crossover probability is set 0.9.

Mutation. Both subtrees mute randomly and the mutaion probability is set 0.5. The chosen crossover and mutation parameters have shown effective in our experiments.

Selection. Tournament selection is applied with size 7.

Others. The population size is set 50, the maximum generation 80, the maximum number of subtree nodes 80. When an individual is evaluated on training set Q with stochastic sampling, a subset containing 1/5 random problem cases of Q is sampled.

4 Empirical Results and Analyses

4.1 Training and Testing Problem Cases

The proposed GP algorithms are run in various environments with different characteristics such as breakdown distribution and duedate tightness. Each environment consists of two problem case sets of the same size, *i.e.,* training set Q and testing set T. Each time the GP is trained on Q to learn high quality PS heuristics and then the evolved program is tested on T for comparisons.

The problems of Q and T are generated randomly in the same way.

- N - five levels are considered (N=15, 25, 40, 60, 80).
- p_i - random number from the discrete uniform distribution *U[1,20].*

	W_1	W_2	W_3	W_4	W_5
B_1M_1	2.0(-93.3)	31.0(-49.8)	54.8(-41.8)	63.8(-49.5)	81.4(-48.7)
B_1M_2	6.2(-93.6)	27.1(-79.2)	46.3(-71.9)	65.8(-66.9)	84.8(-63.6)
B_1M_3	12.6(-94.0)	38.6(-84.4)	56.4(-80.3)	77.3(-76.1)	97.1(-73.2)
B_2M_1	3.6(-59.5)	23.7(-43.1)	42.8(-42.3)	62.4(-41.6)	81.5(-41.6)
B_2M_2	9.2(-66.4)	38.0(-41.1)	58.5(-42.4)	78.1(-43.6)	98.9(-43.7)
B_2M_3	25.3(-53.8)	53.0(-46.0)	75.9(-46.4)	98.0(-47.0)	119.0(-47.9)
B_3M_1	7.2(-44.4)	31.3(-31.0)	52.5(-32.5)	70.7(-35.8)	90.5(-36.6)
B_3M_2	16.2(-46.5)	47.1(-29.6)	68.7(-33.7)	108.3(-22.8)	115.1(-34.9)
B_3M_3	32.6(-47.7)	87.5(-16.1)	120.6(-17.4)	139.3(-25.9)	158.4(-31.1)
D_1	16.4(-73.6)	43.4(-58.1)	69.6(-52.0)	97.6(-47.7)	121.2(-46.8)
D_2	18.4(-68.2)	41.3(-56.4)	62.3(-52.9)	83.3(-50.8)	104.4(-49.5)
D_3	19.5(-67.9)	35.8(-60.9)	54.0(-55.9)	71.8(-53.1)	87.3(-52.5)

Table 1 Values of FT_{GP} and $\Delta r(\%)$

- r_i - random number from the discrete uniform distributions $U[0, \rho N E(p_i)]$, where $E(p_i)$ is the expected processing time and two levels of ρ are included ($\rho = 0.25, 0.50$).
- d_i - $d_i = r_i + \gamma p_i$ where γ follows the continuous uniform distributions *U[a,b]* and three values of (a,b) are considered, i.e., (0,1.2), (1,3) and (3,5) which we refer to as D_1 through D_3. Here, D_1 has the tightest duedates while D_3 the loosest.
- BP - the time between breakdowns is generated from the exponential distributions with mean $\theta E(p_i)$ and three values of θ are considered, i.e., 10, 5, and 2 which we refer to as B_1-B_3. In B_3 situation, breakdowns occur more frequently than in B_1 and B_2.
- MP - the breakdown durations follow some uniform distributions $U[\beta_1 E(p_i), \beta_2 E(p_i)]$ and three pair values of (β_1, β_2) are considered, i.e., (0.1,0.5), (0.5,1.5) and (1,3) which we refer to as M_1-M_3. In the case of (1,3), mean repair time will be higher than in the other two cases. Both BP and MP are anticipated in advance and then keep fixed during the problem instance is solved.

To consider differnet breakdown characteristics, we have nine parameter combinations of $B_i M_j$ $(1 \le i, j \le 3)$. For each combination, there are total $5 \times 2 \times 3 = 30$ subcombinations of the other parameters (N, ρ, γ) and we generate 12 instances for each subcombination. Thus, for each environment $B_i M_j$, there are 360 problem instances for training and 360 ones for testing.

The second class of environments focuses on duedate tightness D_d ($1 \le d \le 3$). There are total $5 \times 2 \times 3 \times 3 = 90$ subcombinations for the remained parameters and we generate 4 instances for each subcombination. Thus the training and testing sets are composed of 360 problem instances respectively in each environment D_d.

Altogether, there are nine environments focusing on breakdown types and three environments on duedate restrictions. For each environments, five values of weight w_T is considered, i.e., 0, 0.25, 0.50, 0.75 and 1.0 which we refer to as W_1-W_5. GP algorithms are trained to produce high quality PS heuristics in each environment with each weight.

4.2 Comparisons with Other Heuristics

In order to test the performance of GP-evolved heuristics, we choose ATC(2)+OSMH heuristics for comparison which have performed well as reported (O'Donovan 1999). For each environment, the objectives of GP-evolved program and ATC(2)+OSMH on testing set T are denoted FT_{GP} and FT_{A2O} respectively. Define the improvement ratio of GP relative to ATC(2)+OSMH as

$$\Delta r = \left(\frac{FT_{GP}}{FT_{A2O}} - 1\right) \times 100\% \qquad (9)$$

where $\Delta r < 0$ means performance improvement of GP over ATC(2)+OSMH. The less Δr is, the better GP shows. The values of FT_{GP} and Δr in all scenarios are illustrated in Table 1 where the numbers outside brackets are FT_{GP} and those inside brackets are Δr.

It can be seen from Table 1 that GP achieves significant improvement over ATC(2)+OSMH in all scenarios ($\Delta r \le -16.1\%$). And it might be interpreted as the flexible integration of sequencing and idle time-inserting mechanisms with bi-tree structures and GP's powerful ability to search in heuristic space. Additionally, this result indicates that man-made or knowledge-based heuristics like ATC(2)+OSMH are not suitable for general scenarios although huge efforts might have been spent on developing them. Thus it is better to find specific heuristics for specific scheduling environments. Machine learning like proposed GP will possibly supply promising ways.

It can also be concluded that in most cases, FT_{GP} degrades as breakdowns occur more frequently (e.g., B_1M_2-B_3M_2 with W_3), repair durations become longer (e.g., B_1M_1-B_1M_3 with W_4) or duedate restricts get tighter (e.g., D_3-D_1 with W_2). Therefore breakdown or duedate characteristics will mainly influence PS performance.

4.3 Roles of Idle time

Just as mentioned above, the inserted idle time serves as the buffer to absorb unpredictable disruptions with shop performance maintained well. To get further perceive about how idle time performs in predictive scheduling, we define the averaged idle time (AIT) of heuristic h on testing set T as

$$\overline{it_T(h)} = \frac{1}{|T|}\sum_{q\in T}\frac{1}{N_q}\sum_{i=1}^{N_q} it_i \qquad (10)$$

Table 2 shows the AIT values of GP-evolved programs in 12 environments. It can be seen that in a certain environment (e.g., B_2M_1), the AIT becomes smaller as weight on tardiness gets larger (W_1-W_5). In the extreme cases where $w_T = 1.0$ (W_5), AIT often approaches *zero.* The reason is that as w_T gets larger, requriements on high shop performance gets stronger and in the case of W_5, the problem becomes robust scheduling and less inserted idle time is preferred. On the other hand, when w_T gets smaller, the planner focuses more on preserving stability and larger AIT is therefore expected to weaken the impact of unavoided failures or contingencies on manufacturing systems.

	W_1	W_2	W_3	W_4	W_5
B_1M_1	0.2	1.0	1.6	0.7	0.2
B_1M_2	0.7	0.4	0.5	0.3	0.0
B_1M_3	1.0	0.0	0.6	0.6	0.0
B_2M_1	0.6	0.3	0.2	0.2	0.0
B_2M_2	1.5	0.1	0.0	0.0	0.0
B_2M_3	1.8	1.0	1.0	0.8	0.3
B_3M_1	0.8	1.0	0.9	0.2	0.0
B_3M_2	3.1	1.0	2.0	6.2	0.0
B_3M_3	8.9	0.0	0.1	0.2	0.0
D_1	1.8	1.3	1.5	0.5	0.1
D_2	0.8	0.8	1.0	0.6	0.0
D_3	0.8	1.1	0.6	0.3	0.0

Table 2 $\overline{it_T}$ Values of GP

4.4 Perspectives of Predictive Scheduling

The discovered programs from GP in various environments can further give us useful guides in design predictive scheduling algorithms. Take the evolved PS heuristic in B_2M_1 with W_5 for example, the simplified subfunctions are expressed by (11) and (12) as follows.

$$pri(i,t) = \left[1 - \frac{2(N-1)}{N_o}\right]p_i - 2d_i \qquad (11)$$

$$itime(i,t) = \min(d_i, N_o, MP - \min(d_i, N_o) - 1) \quad (12)$$

According to (11), those jobs with smaller processing time or duedate will be possible of higher priorities. Hence it seems as the weighted combination of SPT and EDD rules to determine job sequence. What's more, as time passes, the weight on SPT becomes stronger. Thus (11) can be seen as one dynamic combinatorial rule.

In (12), the to-be-inserted idle time is determined by job duedate, breakdown duration and remained job number together. If job duedate or breakdown duration is small, the inserted idle time is also small. Obviously, this conclusion is consistent with our intuition thinking. So GP can learn and reformulate human knowledge in a efficient way.

5 Conclusions and Future Work

In this paper, a GP-based learning system is investigated on single machine predictive scheduling problems subject to stochastic breakdowns. The bi-tree structured chromosomes represent scheduling heuristics in a flexible and nature way and the GP-evolved programs in various environments perform much better than known heuristics. The inserted idle time is proved to be the buffer against uncertain disruptions. The obtained heuristics also supply useful guide for further investigation.

Future work can be carried out on other PS model such as parallel machine or flowshop scheduling problems. The further control strategies are also needed on GP such as parsimony control, multiagent or coevolution model, etc.

Acknowledgments

This work is supported by National Science Foundation of China(Grant No. 60004010) and High-Tech Program of China(2001AA411020).

Bibliography

W. Banzhaf, P. Nordin, R. E. Keller, and F.D. Francone, (1998) "Genetic Programming: an introduction," San Francisco, CA: Morgan Kauffman.

R.W. Conway, W.L. Maxwell, and L.W. Miller, (1967) "Theory of scheduling," Addison-Wesley, Reading, MA.

R. L. Daniels, and P. Kouvelis, (1995) "Robust Scheduling to Hedge against Processing Time Uncertainty in Single Machine Production," Management Science, vol. 41, pp. 363-376.

C. Dimopoulos, and A. M. S., Zalzala., (2001) "Investigating the Use of Genetic Programming for a Classic One-machine Scheduling Problem," *Adv. Eng. Sof.*, vol 32, pp. 489-498.

J.R. Koza, (1992) "Genetic Programming: on the Programming of Computers by Means of Natural Selection," Cambridge, MA: MIT Press.

K. N. McKay, J. A. Buzacott, and F. R. Safayeni, (1989) "The Scheduler's knowledge of uncertainty: The Missing Link in Knowledge Based Production Management Systems," Elsevier Science Publishers.

R. O'Donovan, R. Uzsoy, and K. N., McKAY, (1999) "Predictable Scheduling of a Single Machine with Breakdowns and Sensitive Jobs," *Int. J. Prod. Res.*, vol 37, pp. 4217-4233.

F.A. Roadmmer, and K.P. White, (1988) "A recent survey of production scheduling," IEEE Transactions on Systems, Man, and Cybernetics, vol 18, pp. 841-851.

S. D. Wu, R. H. Storer, and P. C. Chang, (1993) "One Machine Rescheduling Heuristics with Efficiency and Stability as Criteria," *Comput. Ops. Res.*, vol.20, pp. 1-14.