Genetic Programming Heuristics for Multiple Machine Scheduling

Domagoj Jakobović, Leonardo Jelenković, and Leo Budin

University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
{domagoj.jakobovic,leonardo.jelenkovic,leo.budin}@fer.hr

Abstract. In this paper we present a method for creating scheduling heuristics for parallel proportional machine scheduling environment and arbitrary performance criteria. Genetic programming is used to synthesize the priority function which, coupled with an appropriate meta-algorithm for a given environment, forms the priority scheduling heuristic. We show that the procedures derived in this way can perform similarly or better than existing algorithms. Additionally, this approach may be particularly useful for those combinations of scheduling environment and criteria for which there are no adequate scheduling algorithms.

1 Introduction

Scheduling is concerned with the allocation of scarce resources to activities with the objective of optimizing one or more performance measures, which can assume minimization of makespan, job tardiness, number of late jobs etc. Due to inherent problem complexity and variability (most of the real-world scheduling problems are NP complete), a large number of scheduling systems employ heuristic scheduling methods. Given different performance criteria and user requirements, the question arises as to which heuristic to use in a particular environment? The problem of selecting an appropriate scheduling policy is an active area of research [1][2] and a considerable effort is needed to choose or develop the algorithm best suited to the given environment. An answer to this problem may be provided using machine learning methods to create problem specific scheduling algorithms.

The combinatorial nature of most scheduling problems allows the use of search based and enumerative techniques [1], such as genetic algorithms, branch and bound, simulated annealing, tabu search etc. These methods usually offer good quality solutions, but at the cost of a large amount of computational time. Search based techniques are hence not applicable in dynamic or uncertain conditions where there is a need for frequent schedule modification or reaction to changing system requirements (i.e. resource failures or job parameter changes). Scheduling with fast heuristic algorithms is therefore highly effective, and the only feasible solution, in many instances.

In this paper we describe a methodology for evolving scheduling heuristics with genetic programming (GP). Genetic programming has rarely been employed

M. Ebner et al. (Eds.): EuroGP 2007, LNCS 4445, pp. 321–330, 2007.

in scheduling, mainly because it is impractical to use to search the space of potential solutions (i.e. schedules). It is, however, very suitable for searching the space of *algorithms* that provide solution to the problem. Previous work in this area of research includes evolving scheduling policies for the single machine unweighted tardiness problem [3][4][5], single machine scheduling subject to breakdowns [6], classic job shop tardiness scheduling [7][8] and airplane scheduling in air traffic control [9][10]. The scheduling procedure in those papers is however defined only implicitly for a given scheduling environment. In this paper we structure the scheduling algorithm in two components: a meta-algorithm which uses priority values to perform scheduling and a priority function which defines values for different elements of the system. To illustrate this technique we develop scheduling heuristics for multiple proportional machine environment (described in the next section) for which a methodology with GP has, to the best of our knowledge, not been published previously. We also include several combinations of additional requirements, such as dynamic job arrivals, sequence dependent setup times and different scheduling criteria.

2 Parallel Machine Environment

2.1 Problem Statement

In a parallel machine environment, a number n of jobs J_j compete for processing on either of m machines. In a static problem each job is available at time zero, whereas in a dynamic problem each job has a release date r_j. The nominal processing time of the job is p_j and its due date is d_j. Each machine in the system has a speed s_i so that the actual processing time of job j on a machine i is given with $p_{ij} = p_j/s_i$. Relative importance of a job is denoted with its weight w_j. The most widely used scheduling criteria for this environment include weighted tardiness, number of tardy jobs, flowtime and makespan. If C_j denotes the finishing time of job j, then the job tardiness T_j is defined as

$$T_j = \max\{C_j - d_j, 0\} \quad . \tag{1}$$

Lateness of a job U_j is taken to be 1 if a job is late, i.e. if its tardiness is greater than zero, and 0 otherwise. Flowtime of a job is the time the job has spent in the system, i.e. the difference between job release time and completion time: $F_j = C_j - r_j$, whereas the makespan (C_{max}) is the maximum finishing time of all the jobs in a set. Based on these output values, the weighted scheduling criteria are defined as follows: weighted tardiness for a set of jobs is defined as

$$T_w = \sum\nolimits_j w_j T_j \ , \tag{2}$$

weighted number of late jobs as

$$U_w = \sum\nolimits_j w_j U_j \ , \tag{3}$$

and weighted flowtime as

$$F_w = \sum\nolimits_j w_j F_j \ . \tag{4}$$

In the case where a machine may need to process more than one type of job, there is sometimes the need to adjust the machine for the processing of the next job. If the time needed for adjusting depends on the previous and the following job, this is referred to as *sequence dependent setup time* and must be defined for every possible combination of two jobs [11] [12]. This condition further increases the problem complexity for some scheduling criteria.

In the evolution process, a single scheduling criteria can be selected as fitness function where smaller values indicate greater fitness. The total quality estimate of an algorithm is expressed as the sum of criteria values over all the test cases.

2.2 Test Cases Formulation

Each scheduling instance is defined with the following parameters: the number of machines m and their speeds s_i, the number of jobs n, their nominal processing times p_j, due dates, release dates and weights. The values of processing times are generated using uniform, normal and quasi-bimodal probability distributions among the different test cases. The number of jobs varies from 12 to 100 and number of machines from 3 to 20. With machine speeds we can define the effective number of machines $\hat{m}$ as the sum of speeds of all machines:

$$\hat{m} = \sum_{i=1}^{m} s_i \ , \tag{5}$$

where m represents the actual number of machines.

In some of the test environments we allow for the job sequence dependent setup times. A distinct setup time, which does not depend of the speed of the machine, is defined for every possible sequence of two jobs. The values of all of the above parameters are generated in accordance with methods and examples given in [4], [11], [12] and [13]. Overall, we defined 120 test cases for learning and 600 evaluation test cases for comparison of the evolved and existing scheduling heuristics.

2.3 Scheduling Heuristics

The scheduling method investigated in this work is priority scheduling, in which certain elements of the scheduling system are assigned priority values. The choice of the next activity being run on a certain machine is based on their respective priority values. This kind of scheduling algorithm is also called, variously, 'dispatching rule', 'scheduling rule' or just 'heuristic'. The term scheduling rule, in a narrow sense, often represents only the *priority function* which assigns values to elements of the system (jobs in most cases). For instance, a scheduling process may be described with the statement 'scheduling is performed using EDD rule'.

While in most cases the method of assignment of jobs on machines based on priority values is self-evident, in some environments it is not. This is particularly true in dynamic conditions where jobs arrive over time or may not be run before some other job finishes. That is why a *meta-algorithm* must be defined for each scheduling environment, dictating the way activities are scheduled based on their priorities and possible system constraints. The meta-algorithm encapsulates the priority function, but the same meta-algorithm may be used with different priority functions and vice versa [14]. In virtually all the literature on the subject the meta-algorithm part is never explicitly expressed but only presumed implicitly, which can lead to many misunderstandings between different projects.

The time complexity of priority scheduling algorithms depends on the meta-algorithm, but it is in most cases negligible compared to search-based techniques, which allows the use of this method in on-line scheduling [15] and dynamic conditions. All the heuristics presented in this paper, including the evolved ones, provide a solution for several hundred instances in less than a second (since the priority functions are evolved offline).

In this work, we included the following widely used scheduling heuristics for efficiency comparison: weighted shortest processing time (WSPT), earliest due date (EDD), longest processing time (LPT), X-dispatch bottleneck dynamics heuristic [13] (XD), Rachamadugu & Morton heuristic [16] (RM), weighted Montagne heuristic [13] (MON) and Apparent Tardiness Cost with Setups heuristic [11] (ATCS). Each heuristic is defined with its priority function which is used by a meta-algorithm for a given environment (stated in the next section).

3 Scheduling with Genetic Programming

In this work we use the described elements of priority scheduling paradigm, so that the meta-algorithm part is defined manually for a specific scheduling environment and the priority function is evolved with genetic programming using appropriate functional and data structures. This way, using the same meta-algorithm, different scheduling algorithms best suited for the current criteria can be devised. The task of genetic programming is to find such a priority function which would yield the best results considering given meta-algorithm and user requirements. The solution of genetic programming is represented with a single tree that embodies the priority function. After the learning process, single best found priority function is tested on all evaluation test cases and compared with existing heuristics. Following the described priority scheduling procedure, we define the following meta-algorithm which is used with all the existing heuristics as well as with GP evolved priority function for static job availability:

while there are unscheduled jobs **do**
  wait until a machine (k) is ready;
  calculate priorities of all available jobs on machine k;
  schedule job with best (greatest) priority on machine k;
end while

Handling Dynamic Job Arrivals. In a dynamic environment the scheduler can use algorithms designed for a static environment, but two things need to be defined for those heuristics. The first is the subset of the jobs to be taken into consideration for scheduling, since some jobs may arrive in some future moment in time. The second issue is the method of evaluation of jobs which have not yet arrived, i.e. the question should the priority function for those jobs be different and in what way. This can be resolved in the following ways:

1. no inserted idleness - we only consider jobs which are immediately available;
2. inserted idleness - waiting for a job is allowed and waiting time is added to job's processing time in priority calculation;
3. inserted idleness with arbitrary priority - waiting is allowed but the priority function must be defined so that it takes waiting time into account.

When using existing heuristics for comparison, we apply the second approach where necessary, i.e. if the priority function does not take job's release date into account. Genetic programming, on the other hand, is coupled with the third approach, as it has the ability to learn and make use of waiting time information on itself. Scheduling heuristics that presume all the jobs are available are modified so that the processing time of a job includes job's time till arrival (waiting time), denoted with

$$wt_j = \max\{r_j - time, 0\} \quad . \tag{6}$$

Thus, if an algorithm uses only the processing time of a job, that time is increased by wt_j of the job. All the described heuristics, except the XD heuristic which is defined for a dynamic environment, are modified in this manner when solving the dynamic variant of the scheduling problem.

The question remains as to which jobs to include when calculating the priority function? It can be shown that, for any regular scheduling criteria [13], a job should not be scheduled on a machine k if the waiting time for that job is longer than the processing time of the shortest of all currently available unscheduled jobs on that machine (some scheduling software implementations also include this condition [17]). In other words, we may only consider jobs j for which

$$wt_j < \min_i \{p_{ki}\}, \forall i : r_i \leq time \quad . \tag{7}$$

This approach can be illustrated with the following meta-algorithm which is used in dynamic conditions with an arbitrary priority function:

while there are unscheduled jobs **do**
  wait until a machine (k) and at least one job are ready;
  p_{MIN} = processing time of the shortest available job on machine k;
  calculate priorities of all jobs j with $wt_j < p_{MIN}$;
  schedule job with best priority;
end while

Table 1. The genetic programming parameters

Parameter / operator	**Value / description**
population size	10000
max. individual depth	17
selection	steady-state, tournament of size 3
stopping criteria	maximum number of generations (150) or maximum number of consecutive generations without best solution improvement (30)
crossover	85% probability, standard crossover
mutation	standard, swap and shrink mutation, 3% probability each
reproduction	5% probability
initialization	ramped half-and-half, max. depth of 5

Handling Sequence Dependent Setups. Almost any heuristic may be adjusted to include sequence dependant setup time with a method presented in [13]. The job priority obtained with the original function is decreased by a certain value that measures the additional cost brought by setup time for that job. If the original priority value is denoted with π_j, then the priority with setup times is given with

$$\pi_{lj} = \pi_j - \frac{s_{lj}}{(p_{AV}/\hat{m}) \cdot (p_j/s_k)} \ , \tag{8}$$

where l is the last processed job, s_{lj} setup time between job l and job j and p_{AV} the average nominal processing time of all unscheduled jobs. All existing heuristics are modified in this way when solving for setup times, except the ATCS heuristic which is specifically designed for this scheduling condition.

Genetic Programming Parameters, Functions and Terminals. The GP parameters used are presented in Table 1. We did not experiment with many parameter combinations as the GP efficiency did not vary noticably in respect to scheduling heuristic efficiency. The most crucial decision is finding the minimal set of functions and terminals that will satisfy the sufficiency property for a given environment. We define the same function set for every scheduling environment and a different terminal set depending on the variant of the problem (for sequence dependent setups and/or dynamic job arrivals). The complete set, along with guidelines for terminal usage, is given in Table 2.

3.1 Scheduling with Static Job Availability

In a static environment all jobs (and all machines) are available at time zero. The task of genetic programming is to evolve such a priority function that would produce schedules of a good quality for a given performance criteria. We made two sets of experiments: one for the simple static problem and another with additional sequence dependent setups, both optimizing minimum weighted tardiness criteria.

Table 2. The function and terminal set

Function name	**Definition**
ADD, SUB, MUL	binary addition, subtraction and multiplication operators
DIV	protected division: DIV $(a, b) = \begin{cases} 1, \text{if } \lvert b \rvert < 0.000001 \\ a/b, \text{otherwise} \end{cases}$
POS	POS $(a) = \max\{a, 0\}$
Terminal name	**Definition**
Terminals used in every problem variant	
pt	nominal processing time of a job (p_j)
dd	due date (d_j)
w	weight (w_j)
Nr	number of remaining (unscheduled) jobs
SPr	sum of processing times of remaining jobs
SD	sum of due dates of all jobs
SL	positive slack, $\max\{d_j - p_j - time, 0\}$
SLs	positive slack using machine speed, $\max\{d_j - p_j/s_k - time, 0\}$
SPD	speed of the current machine (s_k)
Msm	the sum of all machine speeds (effective number of machines, $\hat{m}$)
Terminals for sequence dependent setups	
STP	setup time from previous to job j
Sav	average setup time from previous (l) to all jobs $\frac{1}{n-1}\sum_{j=1}^{n} s_{lj}$
Terminals for dynamic environment	
AR	job arrival time (waiting time), $\max\{r_j - time, 0\}$

For the first set (notation $Q \,||\, \sum w_j T_j$ in scheduling theory) we conducted 20 runs and achieved mean best result of 37.6 with std. deviation $\sigma = 1.38$ in weighted tardiness as fitness function on evaluation set of 600 unseen test cases. Apart from total criteria values, a good performance measure for a scheduling heuristic may be defined as the percentage of test cases in which the heuristic provided the best achieved result (or the result that is not worse than any other heuristic). This value can be denoted as the dominance percentage. Both types of results are shown in the uppermost section of Table 3 and best results in each category are marked in boldface.

It can be noted that the performance is mainly divided between different heuristics: GP evolved heuristic achieved best weighted tardiness result, WSPT rule best weighted flowtime and LPT best makespan. Another set of 20 runs was conducted in the same environment but with the inclusion of sequence dependent setups (notation: $Q \,|s_{ij}|\, \sum w_j T_j$) and additional GP terminals from Table 2. The best solutions were found with the mean 42.7 and $\sigma = 2.5$. The results for this variant are shown in Table 3. It is clear from the results that in this environment the GP evolved heuristic obtained very good performance over more criteria.

Table 3. Scheduling criteria values and dominance percentages

Heuristic	Scheduling criteria				Dominance percentage			
	Twt	**Uwt**	**Fwt**	**Cmax**	**Twt**	**Uwt**	**Fwt**	**Cmax**
Static job arrivals, weighted tardiness optimization								
GP	**34.1**	28.1	41.1	90.3	**79 %**	21 %	7 %	11 %
RM	46.2	27.4	41.7	92.1	14 %	22 %	1 %	6 %
MON	46.2	**25.0**	36.0	92.9	8 %	27 %	24 %	5 %
WSPT	49.8	25.1	**35.1**	94.4	1 %	**31 %**	**66 %**	3 %
EDD	66.1	36.0	41.3	92.9	4 %	9 %	2 %	6 %
LPT	115.7	44.7	52.6	**83.6**	0 %	0 %	0 %	**73 %**
Static job arrivals, sequence dependent setups, weighted tardiness optimization								
GP	**42.1**	**38.4**	**63.2**	69.4	**87 %**	**75 %**	**72 %**	20 %
ATCS	61.0	44.7	68.7	70.6	9 %	22 %	22 %	9 %
RM	76.4	50.1	78.2	69.7	1 %	12 %	1 %	10 %
WSPT	71.6	47.6	72.9	**66.8**	1 %	15 %	4 %	**45 %**
MON	73.8	49.0	76.7	68.7	2 %	13 %	1 %	15 %
LPT	85.8	52.7	83.2	74.4	1 %	16 %	1 %	2 %
Dynamic job arrivals, weighted tardiness optimization								
GP	**33.0**	**23.9**	26.7	47.9	**78 %**	**45 %**	9 %	11 %
XD	39.3	26.5	28.0	48.5	10 %	18 %	5 %	10 %
MON	39.3	24.6	25.0	48.7	5 %	19 %	29 %	11 %
WSPT	41.1	24.3	**24.5**	48.7	3 %	27 %	**54 %**	9 %
EDD	50.2	33.0	27.6	47.8	6 %	6 %	4 %	13 %
LPT	81.8	39.4	34.9	**44.9**	2 %	4 %	2 %	**70 %**
Dynamic job arrivals, makespan optimization								
GP	78.7	39.1	34.2	**41.9**	5 %	2 %	7 %	**68 %**
XD	**39.3**	26.5	28.0	48.5	**39 %**	24 %	6 %	6 %
MON	**39.3**	24.6	25.0	48.7	33 %	35 %	30 %	8 %
WSPT	41.1	**24.3**	**24.5**	48.7	16 %	**45 %**	**55 %**	6 %
EDD	50.2	33.0	27.6	47.8	9 %	9 %	3 %	8 %
LPT	81.8	39.4	34.9	44.9	2 %	3 %	1 %	40 %
Dynamic job arrivals, sequence dependent setups, weighted tardiness optimization								
GP	**51.1**	**47.5**	**71.7**	87.4	**92%**	**72%**	**86%**	19%
ATCS	67.8	49.8	81.2	91.5	4%	45%	9%	8%
XD	78.1	53.6	87.8	85.7	2%	27%	2%	17%
WSPT	75.7	51.7	84.6	**84.0**	1%	30%	2%	31%
MON	78.3	52.6	87.4	85.7	2%	28%	2%	20%
LPT	81.9	54.2	88.8	85.0	1%	27%	1%	**35%**
Dynamic job arrivals, sequence dependent setups, makespan optimization								
GP	**53.9**	50.2	**65.1**	**73.9**	**89%**	56%	**95%**	**89%**
ATCS	67.8	**49.8**	81.2	91.5	8%	**62%**	3%	7%
XD	78.1	53.6	87.8	85.7	1%	27%	0%	9%
WSPT	75.7	51.7	84.6	84.0	1%	32%	1%	11%
MON	78.3	52.6	87.4	85.7	1%	28%	1%	9%
LPT	81.9	54.2	88.8	85.0	1%	27%	1%	7%

3.2 Scheduling with Dynamic Job Availability

In dynamic environment the jobs have distinct release times, whereas the machines are still available from the time zero. In this variant the heuristics are coupled with the second meta-algorithm and GP terminal set is expanded according to Table 2. We conducted four sets of experiments, two sets without and another two with sequence dependent setups. For each group we experimented with two different scheduling criteria: weighted tardiness and makespan. All sets consisted of 20 runs, out of which the best evolved priority function is compared with existing heuristics on evaluation set of test cases. For the variant without setup times and with weighted tardiness optimization (notation: $Q\,|r_j|\sum w_jT_j$) we achieved mean value of 35.0 with $\sigma = 1.4$. Additional 20 runs are conducted with makespan as GP fitness function (notation: $Q\,|r_j|\,C_{\max}$), for which the mean value was 42.7 with $\sigma = 0.8$; the results for both sets are shown in Table 3.

It can be seen that GP can easily outperform other heuristics for arbitrary scheduling criteria. On the other hand, it is not very likely that a single heuristic will dominate over more than one criteria, which is particularly true for our GP system with single fitness function. If we are after a heuristic with good overall performance, then it is maybe advisable to take some 'general use' existing heuristic, but if we want to maximize efficiency for a single criteria, then the evolved heuristics represent a good choice.

The last two sets of experiments included setup times, and for the first set we conducted 20 runs with weighted tardiness as fitness function (notation: $Q\,|r_j, s_{ij}|\sum w_jT_j$), for which we achieved mean of 52.9 and $\sigma = 1.7$. Finally, 20 runs were conducted with makespan as the performance criteria (notation: $Q\,|r_j, s_{ij}|\,C_{\max}$), and the obtained mean value was 73.9 with $\sigma = 0.34$. The results for both sets are shown in Table 3.

It can be perceived that in the case of a relatively rare scheduling environment, such as dynamic job arrivals and sequence dependent setups, GP evolved heuristic easily outperforms existing algorithms. This may be attributed to the non-existence of appropriate algorithms for this kind of problem, and that is exactly the situation in which this technique offers the most promising use.

4 Conclusion

This paper shows how genetic programming can be used to build scheduling algorithms for multiple machine environment with arbitrary criteria. The scheduling heuristic is divided in two parts: a meta-algorithm, which is defined manually, and a priority function, which is evolved by GP. We defined the appropriate meta-algorithms for static and dynamic variants of the problem, as well as functional and terminal elements which form the GP solution. The results are promising, as for given problems the evolved solutions exhibit better performance on unseen scheduling instances than existing scheduling methods. Heuristics obtained with GP have shown to be particularly efficient in cases where no adequate algorithms exist, and we believe this approach to be of great use in those situations.

References

1. Jones, A., Rabelo, L.C.: Survey of job shop scheduling techniques. Technical report, NISTIR, National Institute of Standards and Technology, Gaithersburg (1998)
2. Walker, S.S., Brennan, R.W., Norrie, D.H.: Holonic job shop scheduling using a multiagent system. IEEE Intelligent Systems (2) (2005) 50
3. Dimopoulos, C., Zalzala, A.: A genetic programming heuristic for the one-machine total tardiness problem. In: Proceedings of the Congress on Evolutionary Computation. Volume 3. (1999)
4. Dimopoulos, C., Zalzala, A.M.S.: Investigating the use of genetic programming for a classic one-machine scheduling problem. Advances in Engineering Software **32**(6) (2001) 489
5. Adams, T.P.: Creation of simple, deadline, and priority scheduling algorithms using genetic programming. In: Genetic Algorithms and Genetic Programming at Stanford 2002. (2002)
6. Yin, W.J., Liu, M., Wu, C.: Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In: Proceedings of the 2003 Congress on Evolutionary Computation CEC2003, IEEE Press (2003) 1050
7. Atlan, B.L., Polack, J.: Learning distributed reactive strategies by genetic programming for the general job shop problem. In: Proceedings 7th annual Florida Artificial Intelligence Research Symposium, IEEE, IEEE Press (1994)
8. Miyashita, K.: Job-shop scheduling with gp. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann (2000) 505
9. Cheng, V., Crawford, L., Menon, P.: Air traffic control using genetic search techniques. In: IEEE International Conference on Control Applications, Hawai'i, IEEE (1999)
10. Hansen, J.V.: Genetic search methods in air traffic control. Computers and Operations Research **31**(3) (2004) 445
11. Lee, Y.H., Bhaskaran, K., Pinedo, M.: A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions **29** (1997) 45–52
12. Lee, S.M., Asllani, A.A.: Job scheduling with dual criteria and sequence-dependent setups: mathematical versus genetic programming. Omega **32**(2) (2004) 145–153
13. Morton, T.E., Pentico, D.W.: Heuristic Scheduling Systems. John Wiley & Sons, Inc. (1993)
14. Jakobovic, D., Budin, L.: Dynamic scheduling with genetic programming. Lecture Notes in Computer Science **3905** (2005) 73
15. Pinedo, M.: Offline deterministic scheduling, stochastic scheduling, and online deterministic scheduling: A comparative overview. In Leung, J.Y.T., ed.: Handbook of Scheduling. Chapman & Hall/CRC (2004)
16. Mohan, R., Rachamadugu, V., Morton, T.E.: Myopic heuristics for the weighted tardiness problem on identical parallel machines. Technical report, The Robotics Institute, Carnegie-Mellon University (1983)
17. Feldman, A., Pinedo, M., Chao, X., Leung, J.: Lekin, flexible job shop scheduling system. http://www.stern.nyu.edu/om/software/lekin/ (2003)