Dynamic Scheduling with Genetic Programming

Domagoj Jakobović and Leo Budin

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

Abstract. This paper investigates the use of genetic programming in automatized synthesis of scheduling heuristics. The applied scheduling technique is priority scheduling, where the next state of the system is determined based on priority values of certain system elements. The evolved solutions are compared with existing scheduling heuristics for single machine dynamic problem and job shop scheduling with bottleneck estimation.

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, a large number of scheduling systems employ heuristic scheduling methods. Among many available heuristic algorithms, the question arises of which heuristic to use in a particular environment, given different performance criteria and user requirements. The problem of selecting the 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 problem at hand. A solution to this problem may be provided using machine learning, genetic programming in particular, 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 etc. These methods usually offer good quality solutions, but at the cost of a large amount of computational time needed to produce such a solution. Furthermore, search based techniques are not applicable in dynamic or uncertain conditions where there is need for frequent schedule modification or reaction to changing system requirements. Scheduling with heuristic algorithms that define only the next state of the system is therefore highly effective in most instances.

Genetic programming has rarely been employed in scheduling, mainly because it is unpractical to use it to search the space of potential solutions (i.e. schedules). It is, however, very suitable for the search of the space of algorithms that provide solution to the problem. Previous work in this area of research includes evolving scheduling policies for 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]. In most cases the authors observe performance comparable to the human-made algorithms. The scheduling procedure 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. This approach allows easier creation of various heuristics in an arbitrary scheduling environment. To illustrate this methodology we address the problem of scheduling with dynamic job arrivals, for which there is a possibility of inserted idleness in resource usage. We also tackle the problem of bottleneck identification in multiple machine environments and define an appropriate algorithm structure for job shop scheduling. The obtained results can be used in a more realistic weighted variant of the presented problems.

2 Priority Scheduling with Genetic Programming

A natural representation for the solution of a scheduling problem is a sequence of activities to be performed on each of the machines. While this representation is most suitable for use in combinatorial optimization, it presents only a solution to the specific scheduling instance, which means that a new solution must be found for different initial conditions. With genetic programming, we have the ability to represent a solution for all the problem instances in a scheduling environment with an algorithm that can be used to generate a schedule.

The scheduling method applied 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 SPT rule'. While in most cases the method of assignment of jobs on machines based on priority values is trivial, 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. This meta-algorithm encapsulates the priority function, but the same meta-algorithm may be used with different priority functions and vice versa. 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 [11] and dynamic conditions (all heuristics presented here provide a solution for several hundred instances in less than a second).

The described structure of the scheduling algorithm allows modular development and the possibility of iterative refinement, which is particularly suitable for machine learning methods. In this work the meta-algorithm part is defined man-

ually for a specific scheduling environment, such as dynamic one machine or job shop. 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.

3 Single Machine Dynamic Scheduling

Problem Statement. In a single machine environment, a number n of jobs J_j are processed on a single resource. 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 processing time of the job is p_j and its due date is d_j. The relative importance of a job is denoted with its weight w_j. In this environment the non-trivial optimization criteria include weighted tardiness and weighted number of late jobs, which are defined as follows: if C_j denotes the finishing time of job j, 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. Weighted tardiness for a set of jobs is defined as

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

and weighted number of late jobs as

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

In the evaluation of scheduling heuristics we use a large number of test cases with different number of jobs, job durations and weights. In order for all the test cases to have a similar influence to the overall quality estimate of an algorithm, we define normalized criteria for each test case. Normalized weighted tardiness is defined as

$$\overline{T_w} = \frac{\sum_{j=1}^{n} w_j T_j}{n \cdot \bar{w} \cdot \bar{p}} \quad , \tag{4}$$

and normalized number of late jobs as

$$\overline{U_w} = \frac{\sum_{j=1}^{n} w_j U_j}{n \cdot \bar{w}} \quad , \tag{5}$$

where n represents the number of jobs in a test case, $\bar{w}$ the average weight and $\bar{p}$ the average duration of all jobs. The average duration is not included in weighted number of late jobs because that criteria does not include any quantity of time dependent on job's processing time. The total quality estimate of an algorithm is expressed as the sum of normalized criteria over all the test cases.

Scheduling Heuristics. 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 it takes waiting time into account.

When using existing heuristics for comparison, we apply the second approach where necessary, i.e. if priority function does not take job's release date into account. The 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 the processing time of a job, that time is increased by wt_j of the job. This modification is not necessary for algorithms that are specifically designed for dynamic conditions, i.e. which already include release date information in priority calculation. It can be shown that, for any regular scheduling criteria [12], a job should not be scheduled if the waiting time for that job is longer than the processing time of the shortest of all currently available unscheduled jobs. In other words, we may only consider jobs j for which

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

This approach may be illustrated with the following meta-algorithm using an arbitrary priority function:

while there are unscheduled jobs **do**
  wait until machine is ready;
  p_{MIN} = the duration of the shortest available job;
  calculate priorities of all jobs with $wt_j < p_{MIN}$;
  schedule job with best priority;
end while

In the above algorithm, 'best' priority may be defined as the one with the greatest or the lowest value, which is purely a matter of definition. For purposes of efficiency comparison we used the following heuristics: weighted shortest processing time (WSPT), earliest due date (EDD), weighted Montagne heuristic [12] (MON), Rachamadugu & Morton heuristic [13] (RM) and X-dispatch bottleneck dynamics heuristic [12] (XD). Each heuristic is defined with its priority

function which is used as described in the above meta-algorithm. All except the XD heuristic, which is the only one designed for dynamic job arrivals, are modified to include job waiting time; WSPT heuristic, for instance, has the priority function

$$\pi_j = w_j/(p_j + wt_j) \ . \tag{8}$$

Test Cases. Each scheduling instance is defined with the following parameters: the number of jobs, their processing times, due dates, release dates and weights. Job durations may take integer values between 1 and 100 and their weights values between 0.01 and 1 in steps of 0.01. The values of processing times are generated using uniform, normal and quasi-bimodal probability distributions among the different test cases. Release times are chosen randomly in the interval

$$r_j \in \left[0,\ \frac{1}{2}\sum_{i=1}^{n} p_i\right] \ . \tag{9}$$

Job due dates are generated using two parameters: T as due date tightness and R as due date range, which both assume values in interval [0,1]. For each test case due dates are generated with uniform distribution in the interval

$$d_j \in \left[r_j + \left(\sum_{i=1}^{n} p_i - r_j\right)\cdot(1 - T - R/2)\,,\ r_j + \left(\sum_{i=1}^{n} p_i - r_j\right)\cdot(1 - T + R/2)\right] \ . \tag{10}$$

Due date tightness parameter represents the expected percentage of late jobs and due date range defines the dispersion of due date values. The numbers of jobs in test cases are 12, 25, 50 and 100 whereas parameters T and R assume values of 0.2, 0.4, 0.6, 0.8 and 1 in various combinations. We define 100 scheduling instances that are used as fitness cases in learning process and additional 600 instances that are used for evaluation purposes only.

Scheduling with Genetic Programming. The task of genetic program is to find a priority function which is best suited for use with given criteria and meta-algorithm. After the learning process, the best found priority function is tested on evaluation test cases. The solution of genetic programming is represented with a single tree that embodies the priority function. The choice of functions and terminals is a crucial step in the overall optimization process since they must allow the program to use all the relevant information and form an efficient solution. The complete set of primitives used as tree elements is presented in Table 1.

Weighted Tardiness Problem. The described genetic programming process can be used for optimization of an arbitrary scheduling criteria, but the most common one for single machine environment is weighted tardiness. Fitness value

Table 1. The function and terminal set for dynamic one machine problem

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**
pt	processing time of a job (p_j)
dd	due date (d_j)
w	weight (w_j)
N	total number of jobs
Nr	number of remaining (unscheduled) jobs
SP	sum of processing times of all jobs
SPr	sum of processing times of remaining jobs
SD	sum of due dates of all jobs
SL	positive slack, $\max\{d_j - p_j - time, 0\}$
AR	job arrival time (waiting time), $\max\{r_j - time, 0\}$

of a genetic program solution is defined as sum of normalized criteria values, defined as (4), over all 100 learning test cases (smaller values are better). The genetic programming parameters are given in Table 2 (we did not perform any additional parameter tuning to show that even with 'common' parameter values good results could be obtained).

Table 2. The genetic programming parameters

Parameter / operator	**Value / description**
population size	10000
selection	steady-state, tournament of size 3
stopping criteria	maximum number of generations (300) or maximum number of consecutive generations without best solution improvement (50)
crossover	85% probability, standard crossover
mutation	standard, swap and shrink mutation, 3% probability for each
reproduction	5% probability
initialization	ramped half-and-half, max. depth of 5

We conducted 20 runs using the defined meta-algorithm and achieved mean best result of 331.0 with standard deviation $\sigma = 1.92$. The overall solution was chosen among best solutions of each run as the one with the best performance

on the unseen set of 600 evaluation test cases. The results in the form of total normalized criteria values are presented in the lefthand side of Table 3 ('Twt' denotes weighted tardiness and 'Uwt' weighted number of tardy jobs). Apart from total criteria values, the performance measure for each heuristic may also be described as the percentage of test cases in which the heuristic achieved the best known result (or the result that is not worse than any other heuristic). This value can be denoted as the dominance percentage, and comparative results for all the heuristics are shown in the righthand side of Table 3.

It can be perceived that the evolved scheduling heuristic achieved the best overall performance for both scheduling criteria. In addition, we performed experiments with weighted number of tardy jobs as the fitness function and, as expected, the evolved algorithm's efficiency for that criteria was improved. At the same time, the performance in regard of weighted tardiness decreased significantly, so we may conclude that optimization with weighted tardiness as performance criteria pays off better considering overall algorithm quality.

Table 3. Normalized criteria and dominance percentages for one machine problem

	Normalized criteria		**Dominance percentage**	
	Twt	Uwt	Twt	Uwt
GP	330.6	188.8	80 %	49 %
XD	389.7	194.1	21 %	30 %
RM	451.7	210.6	9 %	17 %
MON	623.1	216.7	3 %	8 %
WSPT	845.0	201.6	0 %	21 %
EDD	1280.9	440.0	14 %	13 %

4 Job Shop Scheduling

Problem Statement. Job shop scheduling includes running n jobs on m machines where each job has m operations and each operation is to be processed on a specific machine (more general model involves arbitrary number of operations for any job). Duration of one operation of job j on machine i is denoted with p_{ij}. Every machine and job is considered to be available for processing from the beginning. The operations of each job have to be completed in a specific sequence which differs from job to job. In addition to weighted tardiness and number of tardy jobs, another non-trivial and widely used criteria are weighted flowtime and makespan. Normalized weighted flowtime of a set of jobs is defined as

$$\overline{F_w} = \frac{\sum_{j=1}^{n} w_j F_j}{n \cdot \bar{w} \cdot \bar{p}} \ , \tag{11}$$

where F_j equals to the completion time of the last operation of a job, C_j. Normalized makespan is similarly defined as

$$\overline{C_{max}} = \frac{\max\{C_j\}}{n \cdot \bar{p}} \quad . \tag{12}$$

Although the jobs are considered to be available from the time zero, scheduling on a given machine is inherently dynamic because an operation may only be ready at some time in the future (after the completion of the job's previous operation). We therefore modify the processing time of an operation as in the single machine dynamic problem (inserted idleness approach).

Scheduling Heuristics. Job shop priority scheduling involves determining the next operation to be processed on a given machine. The scheduling on a machine may only occur if the machine is available and if either of the following is true: there are operations ready to be processed on that machine or there are operations which will be ready for processing at a known time in future. The latter situation occurs if the previous operation of a job has already started and we know the time it will finish. This procedure can be described with the following meta-algorithm:

while there are unprocessed operations **do**
  wait for a machine with pending operations;
  calculate priorities of all pending operations;
  schedule best priority operation;
  update machine and next job's operation ready time;
end while

The choice of operations considered for scheduling is still restricted to those operations whose waiting time (6) is smaller than the duration of the shortest available operation. In efficiency comparison we used the following job shop heuristics: WSPT, processing time to the total work remaining (WSPT/TWKR), weighted total work remaining (WTWKR), dynamic slack per remaining process time (SLACK/TWKR), COVERT (cost over time) and Rachamadugu & Morton job shop heuristic (RM). Each heuristic is described with its priority function; detailed descriptions of the listed heuristics can be found in [14] and [12].

Test Cases. The operations processing times and job weights are generated randomly as for the one machine environment. Job numbers are 12, 25, 50 and 100 whereas the number of machines takes values of 5, 10, 15 or 20 in a test case. The expected total duration of all the jobs is defined as

$$\hat{p} = \frac{1}{m} \sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij} \quad , \tag{13}$$

and job due dates are generated randomly with parameters T and R in the following interval:

$$d_j \in [\hat{p}\,(1 - T - R/2)\,, \hat{p}\,(1 - T + R/2)] \quad . \tag{14}$$

We define 160 test cases for learning and 320 evaluation test cases, in addition to 80 instances taken from [15], used for evaluation only.

Scheduling with Genetic Programming. As in the single machine case, the solution of genetic programming is a single tree which represents the priority function to be used with defined meta-algorithm. The choice of functions is similar to the previous implementation, but the terminals are radically different, because they must include different information of the system state. The set of functions and terminals is presented in Table 4.

Table 4. The function and terminal set for job shop problem

Function name	**Definition**
ADD, SUB, MUL, DIV, POS	as in Table 1
SQR	protected unary square root: $SQR(a) = \begin{cases} 1, \text{ if} a < 0 \\ \sqrt{a}, \text{ otherwise} \end{cases}$
IFGT	comparison operator: $IFGT\,(a, b, c, d) = \begin{cases} c, \text{if} a > b \\ d, \text{otherwise} \end{cases}$
Terminal name	**Definition**
pt	operation processing time (p_{ij})
dd	job due date (d_j)
w	job weight (w_j)
CLK	current time
AR	operation waiting time: $\max\{r_{ij} - time, 0\}$, where r_{ij} denotes finishing time of the previous operation (before machine i)
NOPr	number of remaining job operations
TWK	total processing time of all operations of a job
TWKr	processing time of remaining operations of a job
PTav	average duration of all the operations on a given machine
HTR	head time ratio: the ratio of the total time the job has been in the system and total duration of job's completed operations

We conducted 20 experiments optimizing weighted tardiness criteria with mean best value over the runs 147.5 and $\sigma = 1.07$. The best solution was compared with the existing scheduling heuristics on the evaluation set of 400 (320 + 80) test cases. The results in normalized criteria values and dominance percentages are shown in Table 5.

Scheduling with Adaptive Heuristic. It has already been shown [8] that the identification of the bottleneck resource, i.e. the resource with substantially higher load, may improve the scheduling process. As it is generally not known in advance which machine could become a bottleneck, we may try and develop a

Table 5. Normalized criteria and dominance percentages for job shop problem

	Normalized criteria				**Dominance percentage**			
	Twt	Uwt	Fwt	Cmax	Twt	Uwt	Fwt	Cmax
GP	146.1	70.9	105.8	133.5	88 %	12 %	73 %	0 %
RM	158.0	68.5	110.0	121.6	5 %	14 %	1 %	5 %
COVERT	179.8	73.4	119.0	118.7	0 %	11 %	0 %	31 %
WSPT	161.6	69.3	107.9	121.7	2 %	13 %	16 %	5 %
SPT/TWKR	195.5	74.6	123.2	118.9	0 %	11 %	0 %	35 %
WTWKR	166.1	68.4	109.0	127.8	4 %	17 %	10 %	4 %
SL/TWKR	225.5	77.0	134.3	123.7	0 %	11 %	0 %	14 %
EDD	206.1	76.5	123.7	128.0	1 %	11 %	0 %	6 %

heuristic to determine such resource on line. We propose a genetic programming approach where there are two distinctive agents, or scheduling heuristics, and GP is responsible for evolving the rule to decide which heuristic is to be applied on a given machine. The solution of genetic programming consists of three parts (represented as trees): the first part, or the decision tree, determines which heuristic should be used at a given moment. The other two parts (scheduling trees) are applied depending on the result of the decision tree. Scheduling trees use the same primitives as in Table 4, but the decision tree should be able to recognize increased load on a machine with appropriate set of terminals. The terminals which can be used in the decision tree are presented in Table 6 (the functions are the same in all trees).

Table 6. The terminal set for decision tree

Terminal name	**Definition**
MTWK	total processing time of all operations on a machine
MTWKr	processing time of all remaining operations on a machine
MTWKav	average duration of all operations on all machines
MNOPr	number of remaining operations on a machine
MNOPw	number of waiting operations on a machine
MUTL	utilization: the ratio of duration of all processed operations on a machine and total elapsed time

As the result of the decision tree is a numeric value, we have to interpret it in some way and define the scheduling process. This procedure can be described with the following meta-algorithm:

for each machine i **do**
  calculate decision tree value (P_i);

end for
while there are unprocessed operations **do**
  wait for a machine with pending operations;
  P_i = decision tree value for current machine;
  if $P_i > P_m$, $\forall m$ **then**
    calculate priorities using the second tree;
  else
    calculate priorities using the first tree;
  end if
  schedule best priority operation;
  update machine and next job's operation ready time;
end while

Using the above adaptive structure, we conducted 20 runs with the same evolution parameters and achieved mean best result of 146.05 and $\sigma = 1.25$. The overall best solution (denoted GP-3) is compared with existing scheduling algorithms and with single tree heuristic (denoted GP). The first row of Table 7 shows the results in comparison with existing heuristics and the bottom two rows compare the described methods (we include only GP values for brevity since the other heuristics are unchanged). The t-test on the results of the methods rejects the null hypothesis with $t = 1.89$ and $p < 0.067$ for normalized criteria and with $t = 4.08$ and $p < 3.34 \times 10^{-3}$ for dominance percentages (when compared to existing heuristics). The difference between the two algorithms in terms of absolute criteria values is not great, which can in part be attributed to the relative proximity to the optimal solution. On the other hand, the relative dominance of the multiple tree algorithm is much greater, as it is able to find non-dominated solution in a majority of problem instances.

Table 7. Performance of single tree (GP) and multiple tree solution (GP-3)

	Normalized criteria				**Dominance percentage**			
	Twt	Uwt	Fwt	Cmax	Twt	Uwt	Fwt	Cmax
GP-3	143.8	67.2	104.5	132.9	94 %	17 %	86 %	0 %
GP	146.1	70.9	105.8	133.5	31 %	11 %	24 %	1 %
GP-3	143.8	67.2	104.5	132.9	64 %	17 %	64 %	0 %

5 Conclusion

This paper shows genetic programming can be used to build scheduling algorithms whose performance is measurable with human-made heuristics for a specific scheduling environment. We addressed the issue of dynamic single machine scheduling for which a suitable meta-algorithm and appropriate data structures

are defined. Additionally, a multiple tree adaptive heuristic is proposed for job shop scheduling problem, where decision tree is used to distinguish between resources based on their load characteristics. The results are promising, as for given problems the evolved heuristics exhibit better performance than existing scheduling methods. The presented methodology can be particularly useful in scheduling environments where there are no adequate algorithms and could alleviate the design of an appropriate scheduling procedure.

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. 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)
12. Morton, T.E., Pentico, D.W.: Heuristic Scheduling Systems. John Wiley & Sons, Inc. (1993)
13. 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)
14. Chang, Y.L., Sueyoshi, T., Sullivan, R.: Ranking dispatching rules by data envelopment analysis in a job shop environment. IIE Transactions **28**(8) (1996) 631
15. Taillard, E.: Scheduling instances. "http://ina.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html" (2003)