ENHANCED PRIORITY RULE SYNTHESIS WITH WAITING CONDITIONS

Andreas Beham[(a)], Monika Kofler[(b)], Stefan Wagner[(c)] , Michael Affenzeller[(d)], Helga Heiss[(e)] , Markus Vorderwinkler[(f)]

[(a – d)] Upper Austria University of Applied Sciences
School for Informatics, Communications, and Media
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, 4232 Hagenberg, Austria

[(e-f)] PROFACTOR GmbH
Im Stadtgut A2
4407 Steyr-Gleink, Austria

[(a)]andreas.beham@fh-hagenberg.at, [(b)]monika.kofler@fh-hagenberg.at, [(c)]stefan.wagner@fh-hagenberg.at,
[(d)]michael.affenzeller@fh-hagenberg.at, [(e)]helga.heiss@profactor.at, [(f)]markus.vorderwinkler@profactor.at

ABSTRACT
This work concerns the automated synthesis of priority rules for schedule optimization. Metaheuristic optimizers, in particular genetic programming (GP), are applied to develop the rule system for several scheduling situations in manufacturing scenarios. In this work, the rules are enhanced with a "no work" decision that leaves the deciding entity in a waiting state. Through simulation experiments it is shown how this enables the rule system to achieve a wider range of solutions.

Keywords: priority-rule, dispatching, scheduling, genetic programming

1. INTRODUCTION

Scheduling is one of the key problems in the manufacturing industry. A bad schedule can result in problems such as low throughput, long lead times, large amounts of work in process (WIP) and failure to meet the shipping deadlines. Given an increasing number of product variety and customizations, naturally, it is difficult not to struggle with any of these. Usually however, companies focus on the last problem of matching the due dates only as they try to create and maintain a positive image in the eyes of their business partners. If there are signs that a deadline cannot be met, extra human effort is added to hold the deadline, sometimes "at all costs". Still, the other problems remain and contribute to the overall situation. For workers this means that they have to work in a more stressful environment with deteriorating motivation.

For a large production site with many jobs, solving a scheduling problem is an arduous task, even for a computer. A scheduling problem such as the job shop scheduling problem (JSSP) is NP-hard which means that there does not exist a polynomial time algorithm that calculates the optimal solution (Pinedo 2001; Garey, Johnson, and Sethi 1976). The complexity of these problems grows exponentially with the problem size, making it especially difficult to provide good solutions for larger and larger problems.

Algorithms that solve such problems can be classified as being either *online* or *offline*, depending on what kind of information is available to them (Albers 1997). Online algorithms schedule immediately while offline algorithms know all jobs to be scheduled in advance.

In this work a priority rule-based scheduler is introduced that is able to delay a certain decision and consider it at a later time. It is not a pure online algorithm as it does not schedule jobs immediately, but is also not offline in that it does not know about all jobs in advance. Rather it aims to bridge the gap between online and offline algorithms. It is trained with several possible scenarios through simulation and thus has some expectations of how the near future may look like. It does not know about all the jobs that are to be scheduled in advance, but it has learned about a possible set of these jobs in the training phase. Thus the rule encodes an expectation of the set of jobs, but does not know the actual set of jobs. Based on this learned knowledge, the algorithm can delay a decision when it looks more promising to make the decision at a later stage.

1.1. Literature Review

Synthesizing priority rules is still a rather young field of research. Existing publications describe the use of machine learning methods such as classifiers to derive new dispatching rules. The training and analysis of such rules is still performed on simple models. In the following a brief overview is given.

(Olafsson and Li 2010) describe a method to learn new scheduling rules from generated schedules by simple rules using data mining techniques. They

combine a decision tree learner and an instance selector to derive high quality decision trees and thus priority rules that describe simple precedence conditions. For any two jobs the decision tree decides which of them should come before the other. A similar approach is described in (Aufenanger, Varnholt, and Dangelmaier 2009), but they learn from the best of a large set of randomly generated schedules with different parameters. Their classifier detects similar situations and applies the same parameters that have led to promising results in the learning phase.

(Mouelhi-Chibani and Pierreval 2010) describe an artificial neural network (ANN) that receives system parameters and states dispatching rules that should be applied as inputs and outputs. They use a simplified flow-shop model on which they run their learning method and compare it with dispatching rules selected by experts. Their approach is able to achieve a similar quality, but without requiring domain expert knowledge.

Another different priority rule base approach is described in (Vázquez-Rodríguez and Petrovic 2009). Similar to (Mouelhi-Chibani and Pierreval 2010) they do not synthesize new rules, but optimize a batch of dispatching rules that are applied in cycles to rank the jobs in the queue. Their dispatching rule based genetic algorithm showed good performance on a number of test problems. However they do not take system or state information into account. This approach is more an example of an offline algorithm. The derived batches are likely not reusable and specific to a certain instance.

The remainder of the paper is organized as follows: We will describe the method in section 2 and describe some of the scenarios that it is applied on in section 3. We will show and analyze the results in section 4 and finally draw conclusions.

2. PRIORITY RULE SYNTHESIS

(Olafsson and Li 2010) mention the interpretability of the learned model that is used as dispatching rule: "The interpretability of the results is also important and this can be directly related to the complexity of the resulting classification algorithm." The model of a decision tree can be well interpreted, but the question remains how powerful it is to include the necessary information for more complex scenarios. The neural network of (Mouelhi-Chibani and Pierreval 2010) cannot be interpreted as easily and can be seen more as an automated black box method, however, ANN might scale to more complex situations more easily.

It is a challenge to have a model class that is expressive enough to describe complex scenarios, but still remain interpretable. From the point of view of the authors genetic programming might be a good solution to this problem as has been shown in (Beham, Winkler, Wagner, and Affenzeller 2008).

2.1. From Simple to Complex Priority Rules

In the case of online algorithms, such as priority-rule based scheduling, with non-preemptive tasks the question on what to do next needs to be determined at fixed points in time, e.g. when a job is introduced into the system, a machine becomes idle, or another kind of event changes the system state. Priority rules generally determine the next action by ranking each possible action according to predefined criteria and choosing the best ranked action. The rule itself is usually not very complex and can be evaluated in very short time on even a large set of possible actions. When an offline algorithm would probably run from several minutes to hours to react to a change in the system, the online algorithm takes just a few seconds.

Several simple priority rules have already been identified in the literature, among them for example earliest due date first, shortest processing time first, least number of steps to complete first, and many others (Panwalkar and Iskander 1977). These rules usually take just a very small number of attributes into account.

In previous publications it was described how these simple priority rules can be combined to complex priority rules using techniques such as genetic algorithms (GA) or genetic programming (Beham, Winkler, Wagner, and Affenzeller 2008; Kofler, Beham, Wagner, and Affenzeller 2009). By combining simple rules with mathematical functions and constants one can include a multitude of different attributes into the rule, see for example Figure 1.

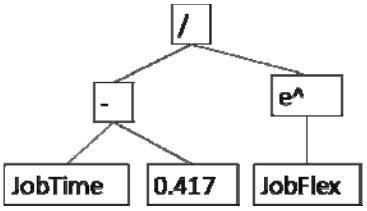

Figure 1 Example of a formula found by GP that combines the simple priority rules JobTime, which is the amount of time spent in the current queue, and JobFlex, the number of alternate machines a job can be processed on.

These complex priority rules are synthesized by combining members of the terminal set and members of the non-terminal set to create a mathematical formula. The terminal set consists of attributes of all entities (jobs, machines, workers, buffers, resources,…) that could possibly be taken into account as well as numerical constants. The non-terminal set consists of mathematical functions and relations.

2.2. Benefit of Waiting Conditions

Even though this optimization procedure can result in quite complex rules that are fit to a certain situation, there is a disadvantage in how these rules are applied. Currently a priority-rule would always rank available actions by the calculated priority and thus take an action at the first possible time. If there is only one job in the queue and the machine becomes idle, an online algorithm generally would schedule that job immediately. This leads to situations where e.g. a job

could be scheduled even though its due date is still far away. Under the presence of sequence dependent setup times this could for example mean that a setup step is introduced.

We thus propose to enhance priority rules by introducing a "wait" decision with the ability to delay a certain decision and reevaluate it at a later time. With this in mind a priority rule will not attempt to take an action at the first possible time, but rather aims to take it at the best possible time given some expectations of the future state.

Naturally, predicting the future is a difficult and error prone task and requires that the priority rule has in some ways learned about it. In the case of genetic programming this ability can be implicitly learned and trained. If the rule is evaluated by simulation in a virtual plant, then those rules that make the best estimation on the future state are able to perform the best decisions at the best time. They will dominate the others during the optimization run. As a result we will obtain an online algorithm in the form of a complex priority rule which has been trained offline. This algorithm is better informed than a pure online algorithm and can thus make better decisions, but of course it is also specific to the scenario or scenarios that it has been trained with. The rule implicitly decides whether the expected future is a more promising time to take a decision than the current time.

To achieve this behavior, we use a threshold level, that is, a certain priority that needs to be surpassed for an action to be considered. Because GP is able to produce any complex priority-rule within any range we can arbitrarily decide on such a threshold, e.g. zero. The action that will finally be selected by the complex priority rule is the one with the highest priority greater than the threshold level. If no action has a priority greater than the threshold, the decision will be delayed until another event occurs that possibly results in a different ranking.

2.3. Genetic Programming

Genetic programming (GP) is a metaheuristic with a formula tree as its solution representation. Unlike the canonical genetic algorithm that uses a binary string to represent a solution of the problem domain, GP uses a mathematical formula which consists of functions (nodes) and terminals (leafs). The set of functions ranges from mathematical operations such as addition, subtraction, multiplication, and division to more complex ones such as cosine or the exponential function. In addition GP is able to make use of logical functions such as IF-THEN-ELSE or the logical connectors AND, OR, and NOT. The terminals in such a function tree are either constant values or variables. In the case of rule synthesis the variables represent the current state of the production system, job characteristics, or the rank that would be obtained with a simple priority rule.

Out of the possible space of function trees genetic programming then creates a random initial population and uses selection, crossover, and mutation to enhance and optimize them over the course of the simulated evolution.

The power to combine these simple priority rules into complex rules and enrich them with even more information on the current state is one of the major advantages of this approach. If GP is given a good set of information on the state of the system and a good set of simple priority rules it is able to combine this information in complex and presumably higher quality rules.

3. MANUFACTURING SCENARIOS

Systems such as the Game of Life show that even simple rules can lead to very complex results in the end. So, to evaluate the behavior of a priority rule it is required to test them in various scenarios. Computer simulation is one technique of evaluating these rules given a model of the production system that it will be applied to.

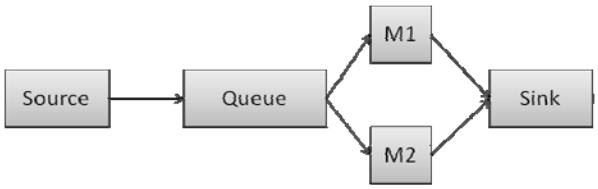

Figure 2 Simplified manufacturing scenario involving two machines, and a shared queue. Products flow along the arrows from source to sink and through either one of the two machines.

To evaluate the enhanced rules we have defined a basic manufacturing scenario which is shown in Figure 2. There is a set of products **P** with two kinds of products p_1 and p_2 and a set of machines **M** with two machines m_1 and m_2. Each product should be painted in shades of gray out of the set **G** at either machine. There are three different shades g_1, g_2, and g_3. There is a processing time matrix **T** with elements t_{ij} that defines the processing times for each $p_i \in \mathbf{P}$ on machine $m_j \in \mathbf{M}$, a cost matrix **C** with elements c_{ij} that defines the costs for producing product $p_i \in \mathbf{P}$ on machine $m_j \in \mathbf{M}$, and a setup matrix **S** with elements s_{kl} that defines the time required to change from shade $k \in \mathbf{G}$ to shade $l \in \mathbf{G}$.

We consider several scenarios where one time the costs should be optimized such that a priority rule is found that assigns each product to the machine with the lowest cost, and another scenario where setup times are taken into account.

3.1. Scenario A

The first scenario is a very simple scenario that primarily exists as a proof of concept. It does not consider setup times, and the only thing that a certain path through the shop has an effect on is costs. Certain products are cheaper to manufacture on certain machines. The optimal rule in this case is known beforehand and selects the appropriate products to be processed on their cheapest machine. Since no other

factors such as due dates are considered in the objective functions the rule is rather simple.

Table 1 Cost matrix of Scenario A

C [€]	p_1	p_2
m_1	1	2
m_2	2	1

The processing time matrix **T** is defined such that $\forall_{i,j} t_{ij} = 5$, the setup time matrix **S** is the zero matrix. The products are generated with equal probability at a rate that is equal to the mean of the processing times of the machines.

There is a difference though in the optimal rule, with regard to total costs, in the case without waiting conditions compared to the case with waiting conditions. Allowing a wait enables the machine to not act on a queue with products that would cost too much to be processed. This rule can achieve a better quality, because it can achieve a perfect split.

However, naturally there are disadvantages. The downside of the rule goes hand in hand with the implication of introducing a decision to not work: Not producing anything would lead to zero costs. This is of course a problem that has to be dealt with and will be discussed in the results section.

3.2. Scenario B

This scenario is slightly more advanced than the previous and also more complex so that an a priori optimal solution is not known. The processing time matrix is given in Table 2 and the setup time matrix is given in Table 3. The cost matrix is the same as in Scenario A and given in Table 1. Note that minimizing the costs as well as the total processing time are in this case conflicting goals. The more expensive process is quicker to complete. As in Scenario A each product has equal probability to enter the queue, and the shade that it should be painted with is also chosen with equal probability among all three choices.

The fitness function has also changed due to the multi-objective nature of the problem. It is a combination of average costs per product and average processing time per product such that both have about equal weight.

Table 2 Matrix of processing times of products on the available machines of Scenario B.

T [min]	p_1	p_2
m_1	6	4
m_2	4	6

Table 3 Matrix of setup times for Scenario B. The value represents the time in minutes to change from the shade in the row header to that in the column header.

S [min]	g_1	g_2	g_3
g_1	0	1	1
g_2	2	0	1
g_3	4	2	0

4. RESULTS

4.1. Software Environment

To test the hypothesis that a waiting condition is beneficial to the synthesis of advanced priority rules a simulation model that describes the simplified manufacturing scenario is created using the simulation framework SiRO (http://www.profactor.at/en/production/produkte/siro.html). It models the flow of products as shown in Figure 2. Products arriving at the *Sink* are evaluated according to the selected performance criteria and after a predefined number of products have been passed through the system the aggregated performance over all finished products is calculated and presented to the GP metaheuristic as fitness value for the priority rule that was used in the simulation.

The GP algorithm was configured and tested in the HeuristicLab open source optimization environment (http://dev.heuristiclab.com). HeuristicLab was first released to the public in 2004. The latest version 3.3 was released in 2010 and aims at providing a unified environment for metaheuristic optimization intended to cover algorithm design, configuration, experimenting, and analysis.

The results were computed using HeuristicLab's genetic programming in version 3.2.0.2683. The full configuration is given in Table 4.

Table 4 Configuration of the genetic programming optimizer

Parameter	**Value**
PopulationSize	100
Selector	TournamentSelector
Tournament Group Size	2
Crossover	Exchange sub trees
Mutators (one of them will be chosen each time it is applied)	OnePointShaker, FullTreeShaker, ChangeNodeTypeManipulation, CutOutNodeManipulation, DeleteSubTreeManipulation, SubstituteSubTreeManipulation
MutationRate	15%
Elites	1
Maximum Generations	1000

4.2. Scenario A

The problem of the "no work" decision appearing very often in certain rules posed a difficulty when evaluating their performance. Some rules exist that did not dispatch any jobs and the machines remained empty during the whole run, meanwhile the queue grew large and the evaluation of several thousand jobs in the queue slowed the simulation down considerably. Also there have been rules that managed to assign only half of the jobs correctly and the other half remained in the queue. These problems required to redesign the fitness function such that it combined two objectives: That of producing at the least cost possible and that of finishing as much jobs as possible. So the fitness function, at the end of a simulation run, assigned each job still in the queue the highest possible production cost. Finally the total costs were divided by the amount of jobs injected into the system. Additionally an early abort criterion was added to the model that detected when a rule would not dispatch any jobs at all. If such an abort occurred, the fitness value was multiplied with the ratio of injected to finished jobs as additional penalty.

For the simple scenario the best result is easy to obtain when **C** is known to the priority rule. The best rule was found quickly and says

$$1.2 * c \leq \frac{2.44}{1.1 * c}$$

where c is substituted by the cost of the currently evaluated product on the currently evaluated machine. This formula can be simplified and interpreted easily: Any product is chosen where the costs are below a certain threshold. In our a priori optimal solution this would be an arbitrary number in the half-open interval]1, 2]. In the concrete rule this is the value 1.36.

4.3. Scenario B

In the more advanced second scenario sequence dependent setup times play an important role as they delay the production process. As has been stated the goal is to find a rule where the machine's utilization can be lowered while maintaining the costs. After several generations, following rule was found to solve this problem best

$$-0.6 * s + 0.8 * e^{1.2*c} * i$$

where again c is the cost, s is the appropriate setup time given the product shade and the current shade of the machine; *i* is the length of the period a machine has remained idle which was an additional system state available to the optimizer. Again with a little work the formula can be easily interpreted: The first job which doesn't require setup should be taken, but if there is setup necessary, wait for a portion of the setup period and then take the job. The term -0.6*s is actually pulling the decision below the threshold line until the factor *i* increases the costs to a level that surpass the setup effort. The synthesized rules are quite elegant to analyze and reveal several things about the underlying system. A rule that has been found to optimize a certain situation can say a lot about that situation also providing important feedback to the human operator. Note that these cases are not free of starvation, because a starving job does not affect the fitness in this case.

A comparison of the actual results reveals that the found rule has slightly higher costs per unit, but a much lower utilization and processing times while having about an equal number of finished units. Table 5 lists the difference between a FIFO rule and the optimized rule broken down to several output values. As can be seen the optimized rule has finished about the same number of units, albeit at a slightly higher cost, but at a much lower utilization that helps to reduce stress in the plant. Certainly the rule could be optimized to favor costs more than processing time by changing the weights in the fitness function since costs and processing time are conflicting goals.

Table 5 Comparison of the FIFO rule with that found by GP

	FIFO	GP
Utilization AP_1/AP_2	96.7%/96.7%	74%/80%
Avg. cost per product	1.51€	1.55€
Processing time	372.75s	297.55s
Produced Units	3135	3129

5. CONCLUSIONS

This work described the synthesis of priority rules using genetic programming and showed the benefit of including a no work decision. When a simulation model of the underlying process is available the rules can be optimized with an expectation of the future implicitly present in the rules themselves. The rules may choose to not perform decisions at the current time, but to delay them if it seems likely that a better choice is soon to arrive.

Naturally a problem with all learning approaches and also with the one described is the characteristics of training data set in comparison to the real situation. If they are very similar the rule has a good chance to make near optimal decisions, however if the real situation differs greatly the decisions will be made under false assumptions. One way to overcome this problem is to continually monitor the process and optimize rules in parallel to the real situation.

ACKNOWLEDGMENTS

The work described in this paper was funded by the Austrian Research Agency (FFG) under grant FdZ-Projekt PROCOMPOSITE: FFG-ProjNr.: 813760 F-WGF.

REFERENCES

Albers, S., 1997. Better bounds for online scheduling. STOC '97: *Proceedings of the 29th annual ACM symposium on Theory of computing*, pp. 130-139.

Aufenanger, M., Varnholt, H., and Dangelmaier, W., 2009. Adaptive Flow Control in Flexible Flow Shop Production Systems - A Knowledge-Based Approach. *Proceedings of the 2009 Winter Simulation Conference*, pp. 2164-2175, Austin, TX, USA.

Beham, A., Winkler, S., Wagner, S., and Affenzeller, M., 2008. A Genetic Programming Approach to Solve Scheduling Problems with Parallel Simulation. *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium (IPDPS08),* IEEE.

Garey, M.R., Johnson, D.S., and Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research,* 1 (2), pp. 117-129.

Kofler, M., Beham, A., Wagner, S., and Affenzeller, M., 2009. Evaluation of Various Dispatching Strategies for the Optimization of a Real Production Plant. *Proceedings of the 2nd International Symposium on Logistics and Industrial Informatics* (LINDI 2009), pp. 25-30. IEEE Publications.

Mouelhi-Chibani, W., and Pierreval, H., 2010. Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, 58, pp. 249-256.

Olafsson, S., and Li, X., 2010. Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, doi:10.1016/j.ijpe.2010.06.004.

Panwalkar, S.S., and Iskander, W., 1977. A Survey of Scheduling Rules. *Operations Research,* 25, pp. 45-61.

Pinedo, M., 2001. Scheduling: Theory, Algorithms and Systems, 2nd edition. Prentice Hall.

Vazquéz-Rodríguez, J.A., and Petrovic, S., 2009. A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, doi: 10.1007/s10732-009-9120-8.

AUTHORS BIOGRAPHY

ANDREAS BEHAM received his MSc in computer science in 2007 from Johannes Kepler University (JKU) Linz, Austria. His research interests include heuristic optimization methods in production environments. Currently he is a research associate at the Research Center Hagenberg of the Upper Austria University of Applied Sciences (Campus Hagenberg).

MONIKA KOFLER studied Medical Software Engineering at the Upper Austrian University of Applied Sciences, Campus Hagenberg, Austria, from which she received her diploma's degree in 2006. She is currently employed as a research associate at the Research Center Hagenberg and pursues her PhD in engineering sciences at the Johannes Kepler University Linz, Austria.

STEFAN WAGNER received his MSc in computer science in 2004 and his PhD in engineering sciences in 2009, both from Johannes Kepler University (JKU) Linz, Austria; he is professor at the Upper Austrian University of Applied Sciences (Campus Hagenberg). Dr. Wagner's research interests include evolutionary computation and heuristic optimization, theory and application of genetic algorithms, machine learning and software development.

MICHAEL AFFENZELLER has published several papers, journal articles and books dealing with theoretical and practical aspects of evolutionary computation, genetic algorithms, and meta-heuristics in general. In 2001 he received his PhD in engineering sciences and in 2004 he received his habilitation in applied systems engineering, both from the Johannes Kepler University of Linz, Austria. Michael Affenzeller is professor at the Upper Austria University of Applied Sciences, Campus Hagenberg, and head of the Josef Ressel Center *Heureka!* at Hagenberg.

HELGA HEISS works for PROFACTOR GmbH and received her bachelor degree in hardware software systems engineering in 2008 and currently pursues her master degree in software engineering at the Upper Austria University of Applied Sciences in Hagenberg. Her research interests include computer simulation and software development.

MARKUS VORDERWINKLER received his diploma and doctoral degrees both in electrical engineering from Vienna University of Technology. Dr. Vorderwinkler works for PROFACTOR GmbH where he is head of consulting & solutions for simulation based design & optimisation of logistics systems. He managed more than 50 industrial and international research projects. Mr. Vorderwinkler is lector at the Upper Austrian University of Applied Sciences. His research interests include manufacturing, digital factory and computer simulation.

The Web-pages of the authors as well as further information about HeuristicLab and related scientific work can be found at http://heal.heuristiclab.com/. The Web-page of PROFACTOR GmbH can be found at http://www.profactor.at.