GENERATION OF DISPATCHING RULES FOR JOB SEQUENCING IN SINGLE-MACHINE ENVIRONMENTS

Johannes Karder[a], Andreas Scheibenpflug[b], Stefan Wagner[c], Michael Affenzeller[d]

[a],[b],[c],[d]Heuristic and Evolutionary Algorithms Laboratory
University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria
[b],[d]Institute for Formal Models and Verification
Johannes Kepler University, Altenbergerstraße 69, 4040 Linz, Austria

[a]jkarder@heuristiclab.com, [b]ascheibe@heuristiclab.com, [c]swagner@heuristiclab.com, [d]maffenze@heuristiclab.com

ABSTRACT
A typical way to schedule a set of jobs is to evaluate and optimize different job sequences and then process them in the best found order. This global optimization approach can be applied for any set of jobs. Unfortunately, optimizing a subset of these jobs requires a new optimization run for this particular subsequence. In this paper we show the generation of dispatching rules that aid job sequencing in single machine environments using genetic programming and delta features. The rules are applied to sets of jobs and yield priorities depending on certain characteristics. These priorities are then used to create job orders dynamically depending on the last executed job. Once generated for specific scenarios, the rules provide on-the-fly sequence generation capability for queued subsets of jobs. Finally, we compare the performance and robustness of the generated rules against the scheduling approach.

Keywords: dispatching rules, sequence optimization, scheduling, single-machine, genetic algorithms, genetic programming

1. INTRODUCTION
We can categorize manufacturing shops into two groups: Those that have a small product catalog, but produce items in very high lot sizes and those that produce in smaller lot sizes and have a large product catalog. A long list of distinct products combined with small lot sizes can lead to volatile production scenarios, because manufacturers offering a wide variety of products might not produce all of them at the same time, but rather work on-demand and only have to produce a certain subset of their product catalog.
This paper focuses on low volume production scenarios where a single machine needs different tools to be set up to process a number of different jobs placed in a process queue. These jobs are a selection from a superset of jobs, namely the product catalog, based on existing orders. New orders and therefore additional jobs may be added to the process queue at any given time. The goal is to minimize production costs in terms of set-up time and engage in sustainable production steering to increase long term competiveness on the market.
To enable live scheduling for known and newly added jobs, an approach that employs dispatching rules (i.e. priority rules) is investigated. We use the concept of delta features to create rules which take job similarities into account. Whenever new jobs are added to the process queue, a predefined dispatching rule can be used to schedule the jobs. Each rule is created by means of genetic programming (Koza 1992) and specifically designed for a certain environment, i.e. a machine that is used to create products from a certain product catalog.
The rest of this paper is structured as follows: In Section 2 we provide insights into others' related work. Section 3 gives a brief definition of the addressed scheduling problem. Our methodology is explained in Section 4. Experiments, including algorithm and problem parameters, are shown in Section 5. The final Sections 6 and 7 show achieved results, as well as drawn conclusions and possible future research topics, respectively.

2. RELATED WORK
During the last centuries, scheduling problems have been widely discussed in literature. Ullman (1975) writes about NP-completeness of different scheduling problems, including the general scheduling problem and single execution time scheduling. Lenstra et al. (1977) classify scheduling problems on single, different and identical machines and experiment with various parameter to influence complexity. Both Davis (1985) and Van Laarhoven et al. (1992) use evolutionary concepts such as genetic algorithms or simulated annealing for problem solving. Cheng et al. (1999) created a tutorial survey of works on various hybrid approaches for job-shop scheduling practices using genetic methods. Dispatching rules itself have also been addressed numerous times, e.g. by Blackstone et. al (1982), Holthaus and Rajendran (1997) or Tay and Ho (2008). Beham et al. (2008) combines automated generation of dispatching rules and parallel simulation to solve scheduling problems. Comparisons between standard scheduling approaches using genetic algorithms

and the application of dispatching rules have been done by Kim et al. (2007), as well as Longo (2012). In both cases, it was concluded that scheduling performs better than dispatching. Previous work in the domain of dispatching rules for job scheduling was also done by Kofler et al. (2009), where sequence-dependent set-up costs were minimized using both scheduling and dispatching concepts. The authors suggested multiple simple priority rules and then used genetic programming to synthesize more complex rules which use different features from the specified jobs. The authors of this paper (Karder et al. 2015) also created a simulator for a specific kind of scheduling problem from scratch by using Sim# (Beham et al. 2014) with custom extensions. Experience gained from that work is used within this paper.

3. PROBLEM DESCRIPTION

The scheduling problem consists of a number of jobs that have to be scheduled on a single machine. Each job has a certain demand for tools that need to be set up inside the machine before the respective job can be processed by an operator. After the job has been processed, all tools are removed from the machine. A tool storage allows the machine to automatically load and unload most of the required tools without operator interaction. Since different jobs require different sets of tools and the storage capacity is limited, the storage layout can only be configured to support a certain number of jobs. Sets of jobs that can be executed with a particular storage configuration are referred to as job batches. When switching from one job batch to another, the operator has to manually alter the storage configuration to contain all required tools for the next batch. The time consumption of all manual tool changes is also included in the set-up time evaluation. The problem can be evaluated by using three different quality criterions:

1. Set-up Time (ST)
2. Batch Count (BC)
3. Manual Tool Change Count (MTCC)

Changing the quality criterion yields different quality values. ST and MTCC are highly correlating, meaning that lower set-up times lead to less manual tool changes and vice versa, whereas a lower set-up time usually also leads to a greater batch count. When a job sequence needs to be evaluated, a new storage layout is generated in the following way: All referenced tools (starting with job 1) are added to the storage as long as possible. If tools from a job cannot be added anymore, this job is used to start a new job batch and the storage will be altered to contain all tools for the new job batch. If replacing tools is not possible, e.g. because of storage constraints, all tools will be removed before the new ones are added.

4. METHODOLOGY

We use HeuristicLab (HL) (Wagner et al. 2014), a paradigm-independent and extensible environment for heuristic optimization, as the underlying software foundation. It features ready-to-use implementations of many different evolutionary algorithms and problem types, including special variants of genetic algorithms and basic problem implementations that can easily be adapted to specific needs. HeuristicLab's plug-in concept allows to easily extend its functionality by implementing new plug-ins based on existing framework elements. Taking advantage of the plug-in system, we introduce a new problem type which employs a simulation model for evaluating the performance of the dispatching rules and uses its own solution encoding and respective algorithm operators to make it compatible with population-based evolutionary algorithms. To evaluate scheduling solutions, we employ a discrete-event simulation that is built upon the Sim# library. It is able to simulate every movement of the machine's main mechanical parts, which allows us to accurately compute the set-up times of the real machine.

Rule generation is done using genetic programming (GP), a concept introduced by John Koza in 1992. GP, which is included in the list of default HL plug-ins, is a method where objects in form of trees are built using evolutionary concepts such as selection, crossover and mutation. In our case, a tree represents a dispatching rule, i.e. a mathematical formula, in form of a syntax tree. This formula consists of different mathematical operations and operands, including variables and constants. An example of a simple tree is depicted in Figure 1.

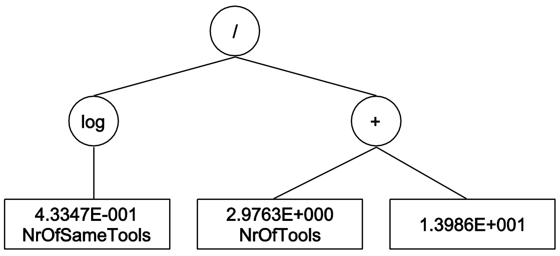

Figure 1: A random GP tree.

The size of a tree is traditionally limited by a maximum length and depth. The tree length is the total number of nodes and the depth is the number of edges from a node to the root node. The tree shown in Figure 1 has a length of 6 and a maximum depth of 2.

To evaluate formulas containing variables, it is necessary to supply a dataset which defines values for these variables. These values are also called features and a set of features supplied to a formula is called row or sample. Trees are therefore evaluated for rows in datasets. Our dataset contains one row for each job from the pool. Some features can be calculated immediately for each job. All other features – we call them delta features – must be calculated on-the-fly, because their values are state-dependent. All delta features are in a way similarity measures, which allow the rule to prioritize according to the current state. Each rule is applied m times to a random subset of n jobs from the pool. Out of this subset, one job is selected to be the first job processed by the machine. For all remaining (i.e. unprocessed) jobs of the subset, the delta features to the currently processed job are calculated. Examples of such features would be the number of same tools, the number of same tool types or the number of same tool kinds the two jobs share. With these additional features, the rule can be used to calculate priorities for the remaining jobs. The job with the highest priority is selected to be processed next, the current job is updated and another iteration is executed until all jobs have been processed. The qualities of the resulting job sequence are calculated according to the selected quality criterion and summed up. The average quality – which should be minimized – makes up the quality of the rule. Pseudocode for this procedure can be seen in Listing 1.

```
r := rule
r_q := 0.0   // rule's summed quality
P := product catalog   // all available jobs
repeat m times
  shuffle P
  J := first n elements from P   // orders
  S := empty list of jobs
  c := first element from J   // current job
  S.add(c)
  repeat n - 1 times
    R := J \ S
    calc. delta features from jobs in R to c
    b := best job from R according to r
    S.add(b)
    c := b
  endrepeat
  q := quality criterion for job sequence def. by S
  r_q := r_q + q
endrepeat
return r_q / m   // rule's average quality
```

Listing 1: Pseudocode describing the evaluation of a single dispatching rule.

Consequently, the dispatching rule is trained to work for a certain product catalog. Subsets are used to simulate a daily or weekly amount of orders. It is important not to train with the whole catalog of products, since there will most likely never be an order containing every product and overfitting to the complete catalog itself should be avoided. Instead, more diverse batches of jobs should be used to mix different smaller orders and create a more general rule for the specific scenario.

5. EXPERIMENTS

To construct an exemplary production environment, we create a product catalog that contains 34 different jobs. These jobs are used to generate a dispatching rule. The evaluation parameters are set as follows:

$$m = 5 \text{ (sample size)}$$
$$n = 18 \text{ (group size)}$$

Additionally, 4 random subsets of jobs containing 10, 15, 20 and 25 jobs are created to reflect daily or weekly orders. The jobs are then scheduled by applying the best found rule. Afterwards, the same subsets are optimized with a standard scheduling approach using a combinatorial optimization problem, which is optimized by a genetic algorithm with offspring selection (OSGA) (Affenzeller and Wagner 2005). We then compare the best found job sequence with the sequence suggested by the rule. All problem instances for the standard scheduling approach and the dispatching approach have been configured as shown in Table 1.

Table 1: Scheduling and dispatching problem parameters.

Parameter	Value
Scheduling	
Optimization Target	Job Order
Quality Criterion	Set-up Time
Dispatching	
Allowed GP Symbols	Add, Sub, Mul, And, Or, Not, Xor, GreaterThan, LessThan, IfThenElse, Variable, Constant
Max. Tree Depth	17
Max. Tree Length	100
Sample Size	5
Group Size	18
Quality Criterion	Set-up Time

The OSGA configurations were similar for the both approaches. Multi-operators were used to induce different operator behaviors into the corresponding optimization procedure. Constant comparison factors were set to work with the concept of weak offspring selection, which means that children are included in the next population if they outperform their weaker parent. For the dispatching approach, the mutation rate was increased by 5 % and gender-specific selection was replaced with proportional selection. A list of algorithm parameters is shown in Table 2.

Table 2: OSGA parameters for the scheduling (1) and dispatching (2) approach.

Parameter	Value
Lower Comp. Factor	0
Comp. Factor Modifier	–
Crossover	Multiple$^{(1)}$ SubtreeSwapping$^{(2)}$
Elites	1
Max. Generations	1000
Max. Sel. Pressure	100
Mutation Probability	10 %$^{(1)}$ 15 %$^{(2)}$
Mutator	Multiple
OS Before Mutation	False
Population Size	100
Selector	Gender-specific$^{(1)}$ Proportional$^{(2)}$
Success Ratio	1

All experiments have been conducted on a distributed optimization cluster (HeuristicLab Hive) with 10 repetitions to produce valid empirical test results.

6. RESULTS

Table 3 shows a direct comparison between the scheduling and dispatching approaches. The scheduling results are the set-up times of the best found job sequences. The dispatching results are the set-up times of the sequences that are generated by the best dispatching rules found for each job set.

Table 3: Test result comparison between scheduling and dispatching approach.

Job Set	Set-up Time [min]	
	Scheduling	Dispatching
js1 (10 jobs)	38.15	39.35 (+3.15 %)
js2 (15 jobs)	55.45	59.14 (+6.91 %)
js3 (20 jobs)	56.03	62.59 (+12.07 %)
js4 (25 jobs)	92.73	103.42 (+11.74 %)

As already described in previous applications (Kim et al. 2007, Longo 2012), the experiments show that the standard scheduling approach achieves better results than the dispatching rules.
By analyzing the results of the executed runs, it appears that the dispatching approach needs more adjustments to produce more robust rules. In particular, we have a look at the 10 best rules that were generated within the 10 repetitions, i.e. each best rule of each run. Figure 2 shows a box-plot of these rules' achieved set-up times, applied to the 4 different job sets. Table 4 lists more detailed measures.

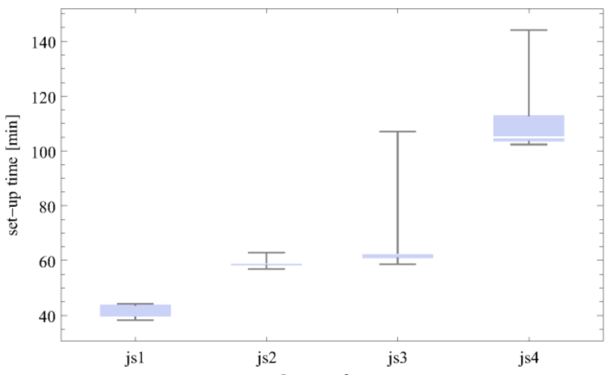

Figure 2: Box plots of set-up times.

Table 4: Statistical overview of the achieved set-up times for each job set.

Measure	Set-up Time [min]			
	js1	js2	js3	js4
Min	38.35	56.99	58.69	102.34
Max	44.26	62.91	107.06	144.12
Med	40.11	59.12	62.69	105.96
Avg	41.22	59.14	66.29	113.31
SD	2.50	1.59	14.41	16.51
Var	6.23	2.52	207.51	272.54
Q1	39.35	58.37	60.86	103.55
Q3	43.89	59.24	62.81	111.71

The rules are not totally stable, as the resulting set-up times have a standard deviation between approximately 1.5 and 16.5 minutes, depending on the job set.
The average runtime of the standard scheduling runs was around 3.5 hours with a minimum of 1.5 hours and a maximum of 4.5 hours. Approximately the same time was consumed on average by the runs of dispatching approach. The execution time of an OSGA run can vary because normally the number of generations is not directly limited by the algorithm parameters itself, but indirectly through the performance of the algorithm during offspring creation.

7. CONCLUSION

We were able to generate rules that enable live scheduling and therefore speed up the scheduling process itself. By applying a predefined rule, respective schedules can be calculated immediately, whereas the standard scheduling approach must be executed separately for each set of jobs. It is also possible to generate multiple rules for a specific environment, which can then be applied to a subset of jobs to generate alternative sequences.
Looking at the results, it can be seen that the dispatching approach cannot reach the quality levels of the standard scheduling approach. This is expected and can be explained by two facts. First of all, the scheduling approach is purely designed to evaluate many different sequences of the respective jobs, which in the end leads to good scheduling qualities. The dispatching approach is designed to evaluate many different dispatching rules. Two distinct rules can however yield the same job sequence. To get n different job sequences, at least n different rules have to be generated. Secondly, the scheduling approach's goal is to yield the best sequence for a specific subset of jobs, whereas the dispatching approach's goal is to yield a general rule that is able create a good schedule for any arbitrary subset of jobs from the product catalog. Additionally, the dispatching approach should be tuned to produce more robust rules. This could be done by e.g. changing the allowed GP symbols or identifying more significant features. Both approaches consume a similar amount of runtime when executed. The runtime-limiting factor is the OSGA's ability to produce better offspring.
Further research includes the generation of multiple dispatching rules for a single product catalog and the creation of a rule portfolio. This portfolio would add an additional layer of optimization in which either multiple rules are applied to create different job sequences to choose from, or additional techniques for on-the-fly rule selection could be implemented.

ACKNOWLEDGMENTS

The work described in this paper was conducted within the NPS (Sustainable Production Steering) project and funded by the Austrian Research Promotion Agency (FFG).

REFERENCES

Affenzeller M., Wagner S., 2005. Offspring Selection: A New Self-Adaptive Selection Scheme for Genetic Algorithms. Proceedings of the 7th International Conference on Adaptive and Natural Computing

Algorithms, pp.218-221. 21-23 March 2005, Coimbra, Portugal

Beham A., Winkler S., Wagner S., Affenzeller M., 2008. A Genetic Programming Approach to Solve Scheduling Problems with Parallel Simulation. Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing, pp.1-5. 14-18 April 2008, Miami, USA

Blackstone J.H., Phillips D.T., Hogg G.L., 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research, 20(1):pp.27-45

Cheng R., Gen M., Tsujimura Y., 1999. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Computers & Industrial Engineering, 36(2):pp.343-364

Davis L., 1985, July. Job Shop Scheduling with Genetic Algorithms. Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pp.136-140. 24-26 July 1985, Pittsburgh, USA

Holthaus O., Rajendran C., 1997. Efficient dispatching rules for scheduling in a job shop. International Journal of Production Economics, 48(1):pp.87-105

Kim I., Watada J., Shigaki I., 2008. A comparison of dispatching rules and genetic algorithms for job shop schedules of standard hydraulic cylinders. Soft Computing, 12(2):pp.121-128

Kofler M., Wagner S., Beham A., Kronberger G., Affenzeller M., 2009. Priority Rule Generation with a Genetic Algorithm to Minimize Sequence Dependent Setup Costs. In: Moreno-Díaz R., Pichler F., Quesada-Arencibia A., eds. Computer Aided Systems Theory - EUROCAST 2009. Heidelberg, Germany:Springer, pp.817-824

Koza J.R., 1992. Genetic programming: On the Programming of Computers by Means of Natural Selection. Cambridge, USA:The MIT Press

Lenstra J.K., Kan A.R., Brucker P., 1977. Complexity of Machine Scheduling Problems. Annals of Discrete Mathematics, 1:pp.343-362

Longo F., 2012. On the short period production planning in industrial plants: a real case study. International Journal of Simulation and Process Modelling, 8(1):pp.17-28

Tay J.C., Ho N.B., 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering, 54(3):pp.453-473

Ullman J.D., 1975. NP-Complete Scheduling Problems. Journal of Computer and System Sciences, 10(3):pp.384-393

Van Laarhoven P.J., Aarts E.H., Lenstra, J.K., 1992. Job Shop Scheduling by Simulated Annealing. Operations Research, 40(1):pp.113-125

Wagner S., Kronberger G., Beham A., Kommenda M., Scheibenpflug A., Pitzer E., Vonolfen S., Kofler M., Winkler S., Dorfer V., Affenzeller M., 2014. Architecture and Design of the HeuristicLab Optimization Environment. In: Klempous R., Nikodem, J., Jacak W., Chaczko Z., eds. Advanced Methods and Applications in Computational Intelligence. Cham, Switzerland:Springer, pp.197-261

AUTHORS BIOGRAPHIES

JOHANNES KARDER received his Master in software engineering in 2014 from the University of Applied Sciences Upper Austria and is a research associate at the Research Center Hagenberg. His research interests include algorithm theory and development as well as production planning and logistics optimization. He is a member of the HeuristicLab architects team.

ANDREAS SCHEIBENPFLUG received his Master in software engineering in 2011 from the University of Applied Sciences Upper Austria and is a research associate at the Research Center Hagenberg. His research interests include parallel and distributed computing. He is a member of the HeuristicLab architects team.

STEFAN WAGNER received his PhD in technical sciences in 2009 from the Johannes Kepler University Linz, Austria. He is a professor at the University of Applied Sciences Upper Austria, Hagenberg Campus. He is the project manager and head developer of the HeuristicLab optimization environment.

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 University of Applied Sciences Upper Austria, Hagenberg Campus.