Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, and E. Zhou, eds.

EVOLVING CONTROL RULES FOR A DUAL-CONSTRAINED JOB SCHEDULING SCENARIO

Jürgen Branke

Warwick Business School
Scarman Road
Coventry, CV4 7AL, UK

Matthew J. Groves

University Of Warwick
Scarman Road
Coventry, CV4 7AL, UK

Torsten Hildebrandt

jasima solutions UG
Wachmannstr. 105
Bremen, Germany

ABSTRACT

Dispatching rules are often used for scheduling in semiconductor manufacturing due to the complexity and stochasticity of the problem. In the past, simulation-based Genetic Programming has been shown to be a powerful tool to automate the time-consuming and expensive process of designing such rules. However, the scheduling problems considered were usually only constrained by the capacity of the machines. In this paper, we extend this idea to dual-constrained flow shop scheduling, with machines and operators for loading and unloading to be scheduled simultaneously. We show empirically on a small test problem with parallel workstations, re-entrant flows and dynamic stochastic job arrival that the approach is able to generate dispatching rules that perform significantly better than benchmark rules from the literature.

1 INTRODUCTION

Scheduling is concerned with the allocation of resources to tasks over a period of time. It is an area of considerable importance to the semiconductor production industry, where manufacturers seek to allocate jobs to machines as efficiently as possible, e.g. to maximise their throughput, or to minimise lateness. In the real world, these problems can be highly complex, (in general they are NP-hard Garey, Johnson, and Sethi 1976) and so the calculation of optimal solutions is impractical, except in small examples. Furthermore, these problems are often stochastic and dynamically changing, e.g. due to random job arrivals, machine breakdowns, stochastic processing times or stochastic rework. To cope with these events, schedulers in practice often resort to dispatching rules. These are simple heuristics that assign a priority to jobs available for processing at a machine, and select the one of highest priority to be processed next whenever a machine becomes idle. Typical examples include First In System First Served (FSFS) or Shortest Processing Time (SPT). Dispatching rules benefit from being decentralised, and make decisions based on local information at the last possible time. That way, they always take into account the latest information, and can naturally cope with stochastic and dynamic environments. However, they are unlikely to provide an optimal solution and may lack the specificity to address the particular intricacies of a problem.

The challenge is therefore to develop dispatching rules customised to a particular scheduling environment. Traditionally, this would have to be done through trial and error (Geiger, Uzsoy, and Aytuğ 2006), whereby a rule would be repeatedly adjusted by an expert and tested through simulation. However, this is obviously

extremely time-consuming. Various approaches, known as hyper-heuristics (Burke et al. 2010), have been proposed to automate this process. These are usually metaheuristics such as Evolutionary Algorithms (EAs) that search over a space of heuristics. Various papers have shown that in particular Genetic Programming (GP) can be successfully used to generate dispatching rules for scheduling scenarios that significantly out-perform manually developed benchmark rules (Branke and Hildebrandt 2015, Pickardt et al. 2010, Pickardt et al. 2013, Geiger et al. 2006).

Most of this research has so far been concerned with investigating problems where the only constraint is machine capacity. In those cases, it is natural to associate a dispatching rule with each machine, and each machine can then independently "select" the next operation using its dispatching rule. In this paper, we extend the scope to scheduling scenarios where production is constrained not only by the availability of machines, but also by operators responsible for loading and unloading jobs. In this case, machine and operator actions need to be coordinated. We propose a way to use dispatching rules in such a scenario, and empirically show that with this approach, again GP is able to generate dispatching rules that outperform benchmark rules from the literature.

The paper proceeds as follows: Section 2 gives a short literature review of the use of hyper heuristics in scheduling and of job scheduling of dual-constrained problems. In Section 3, we provide a description of the flow shop problem. In Section 4, we detail our methodology and the algorithm design of the GP. Empirical results are reported in Section 5 and the paper concludes in Section 6 with a summary and some ideas for future work.

2 RELATED WORK

Production scheduling and the design and testing of dispatching rules has been an active area of research for many years. Haupt (1989) provides definitions of many of the standard heuristics along with some comparison of rules and discussion of various simulation scenarios. The use of hyper-heuristics to automatically generate dispatching rules is more recent, and a comprehensive review has been compiled by Branke et al. (2015), describing various different learning methods, attributes and representations that can be incorporated into a hyper-heuristic. In particular, it highlights the ability of hyper-heuristics to tackle dynamic and stochastic scheduling problems.

The success of Genetic Programming to generate dispatching rules better than human experts is confirmed by a number of studies. Geiger, Uzsoy, and Aytuğ (2006) compare a GP hyper-heuristic approach to well-established dispatching rules for a variety of single machine problems. In another paper Geiger and Uzsoy (2008), also apply a GP hyper-heuristic to a single machine batch processor model, whereby the scheduling procedure must perform two steps, both grouping the waiting jobs into batches and ordering the batches for processing. Yin, Liu, and Wu (2003), also apply GP to a single machine scheduling problem, but include stochastic machine breakdowns. They show empirically that GP can develop high quality heuristics in a range of uncertain environments. Nguyen et al. (2013) compare the performance of evolved rules from three different GP algorithms to well known dispatching rules for a static scenario, finding that the evolved rules behave well on the static problem, but don't deal well with dynamic scenarios. In another paper Nguyen et al. (2012), use a multi-objective GP algorithm to evolve scheduling rules for a dynamic scenario that incorporates due-date assignments, showing that these rules outperform non-evolved scheduling policies that combine dispatching rules with due-date assignment rules. Park, Nguyen, Zhang, and Johnson (2013), also apply a GP framework with embedded heuristic search mechanisms to an order acceptance and sequencing (OAS) problem. This problem considers both acceptance and scheduling decisions simultaneously for a given set of jobs on a single machine. They show experimentally that their GP algorithm can produce competitive heuristics in the field of OAS problems. On more complex manufacturing systems, Pickardt et al. (2010) apply a GP algorithm to generate different rules for different machines simultaneously. Recently, Hildebrandt and Branke (2015), considered methods to accelerate evolutionary algorithms, proposing a new way to combine GP with surrogate models for fitness evaluations. They demonstrate improvements in the convergence speed and solution quality of a simulation-based GP on a dynamic job shop scenario.

Most scheduling research to date has been on the single-constraint problem, where machines are the only limiting factor for production. The dual-constrained problem, where an operator is needed to load or unload an operation at a machine is less well studied, although being a more accurate representation for many real-world problems. For dual-constrained problems, the challenge is how to integrate the operator into the dispatching rule based framework. Clearly, the operator can not just be considered as an additional "machine" type, as operators need to attend to machines, not jobs.

ElMaraghy, Vishvas, and Abdullah (2000) propose an EA and compared its performance in conjunction with various dispatching rules in both dual-constrained and single-constrained static scenarios. They incorporate machine operators by individually assigning a machine and worker to each required operation prior to simulating each scenario. The list of operations with corresponding machines and workers constitute chromosomes in their genetic population. Chaudhry and Drake (2009) also apply an EA to a static dual-constrained problem with parallel machines. However, their scenario includes only jobs consisting of a single operation. In their problem, operation processing times at a machine are dependent on the number of operators assigned to that machine, which is fixed and determined during initialization. They therefore attempt to optimise allocation of both jobs and operators to machines. (Scholz-Reiter, Heger, and Hildebrandt 2009) analyse the performance of different dispatching rules on the dual-constrained "MiniFab" flow-shop scenario that we also use and which includes parallel machine groups and re-entrant flows. In their experiments, the machines and operators decide independently in a two-stage process: Whenever a machine is idle, it decides on the next job to be loaded or unloaded, and sends a corresponding request to the operator. Then, the operator selects from the requests it has received.

3 GENETIC PROGRAMMING FOR DUAL-CONSTRAINED PROBLEMS

3.1 Integrating machine and operator decisions

We propose a new way to integrate machine and operator decisions. In this approach, operators have information on the state of every machine and of every job waiting in the system, both in machine queues and completed jobs waiting to be unloaded from a particular machine. Whenever an operator becomes available, they consider all jobs in the system and determine the job with the highest priority based on their dispatching rule. If this job is ready to be loaded on or unloaded from a machine, the operator performs this operation. In case where the highest priority job is waiting in a queue of a fully loaded workstation, they perform the unloading operation required to clear the machine, before rechecking available jobs. The machines do not have dispatching rules in this case, all the decisions are made by the operators.

3.2 Genetic Programming

The process followed by the GP is outlined in Figure 1. At the start of simulation, an initial population of dispatching rules is randomly generated and evaluated. Individuals are then selected by a tournament process, whereby a fixed number of individuals are randomly selected and their fitness values compared to choose the best. Crossover and mutation operations are then performed to produce a new generation. This process is repeated for a set number of generations before the best individual heuristic is selected as the final output. The parameters used by the GP algorithm can be seen in Table 1.

Individuals in our GP population take the form of trees (see Figure 2), with leaf nodes, or terminals, containing attributes of the job or scenario that can be used to determine scheduling priority, and non-leaf nodes containing functions for combining terminals. As discussed by Branke et al. (2015), it is important to select terminals that are as relevant to the scenario as possible, in order to allow the GP to evolve well-performing rules. For example, common terminals such as Due Date or Slack (time until due date minus remaining processing time) are unlikely to be helpful for minimising flowtime, whereas terminals measuring how long the worker will have to spend performing a loading or unloading task could be very important in allocating the worker's time effectively. It is also important to keep the terminal set compact

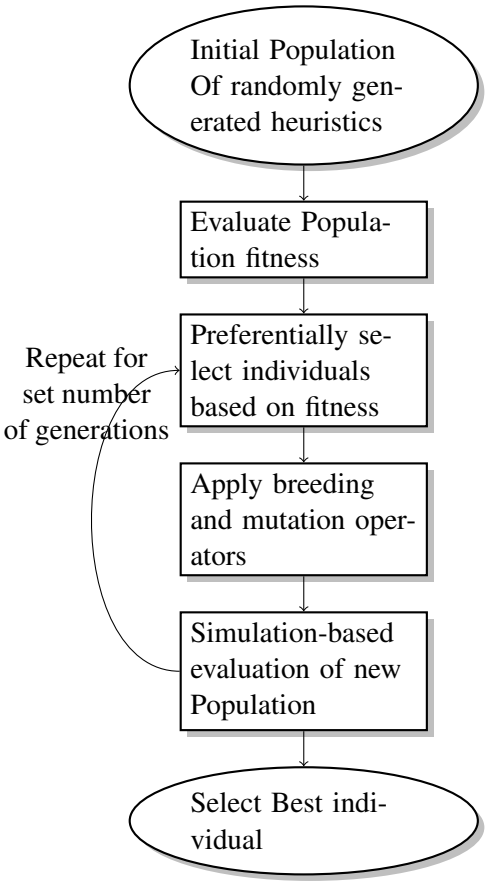

Figure 1: Schematic of the GP algorithm used.

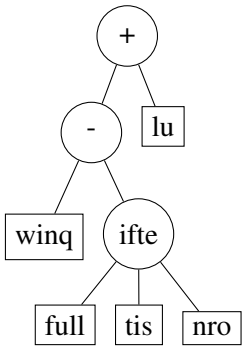

Figure 2: A sample GP tree produced using our function and terminal set.

so that the search space of the GP does not grow too large. We performed an initial series of trial simulation runs, with different terminal set configurations in order to maximise evolved rule performance.

The finalized terminal set consisted of:

- **Number Of Operations Remaining**. Contains the number of operations remaining in a job before completion.
- **PT**. The Processing Time of a job's next operation. This was calculated as the total time required for the current operation to be completed, including loading and unloading times.
- **WINQ**. Work In Next Queue. Used to estimate the amount of time a job will wait in its next queue, based on the sum of the processing times of all jobs in that queue. Lower estimated waits are assigned higher priority.
- **L/U**. Specifies whether the job requires a Loading or Unloading operation. It will return 1 if the job requires loading and 0 otherwise.
- **L/U Time**. Specifies the length of time that the operator will have to spend either loading or unloading the job.
- **F/E**. Full/Empty, whether the workstation that the operation will interact currently has an available machine to be used. It returns 1 if there is an available machine and 0 otherwise.
- **WIQ.** Work In (current) Queue. Contains the total processing times of all the jobs currently waiting in the same queue.
- **TIS**. Time In System contains the time that a job arrived at the job shop.

Table 1: Parameter settings for the GP hyper-heuristic.

Parameter	Value
Generations	30
Population size	500
Initial Population	Randomly generated trees, min depth 2, max depth 6.
Selection	Tournament (size 7)
Max. depth	17
Mutation Probability	10%
Elitism	1%
Terminal set	Number of Remaining Ops (nro), Processing Time (pt), Work In Next Queue (winq), Loading/Unloading (lu), Work In Queue (wiq) Load/Unload Time (lut) Machine status (full) Time In System (tis) Constant value (1)
Function set	$+, -, \times, \div, max, min, ifte$

We used a fairly typical non-terminal function set (as in Hildebrandt, Heger, and Scholz-Reiter 2010) consisting of the basic arithmetic operations of addition, subtraction, multiplication and division along with maximum and minimum functions and a ternary if-then-else function (if $a \geq 0$ then b, else c).

Hildebrandt, Heger, and Scholz-Reiter (2010) consider the best methods for balancing the cost of simulation runs with the quality of heuristic produced when running GP simulations of job-scheduling problems. They suggest that, provided the GP is allowed to run for enough generations, it is more efficient to allocate resources to simulations with a single training evaluation for each individual dispatching rule in each generation, with the random seed used for evaluations changed each generation to reduce over-fitting. We also follow this approach in the current work. We used a population size of 500 and 30 generations to produce our output heuristics, with training evaluations replicating a 5 year time period of the Mini-Fab scenario.

Our GP was implemented in Java and using the ECJ library.

4 PROBLEM DESCRIPTION AND BENCHMARK RULES

4.1 Problem description

The problem we use for empirical evaluation has been taken from Scholz-Reiter, Heger, and Hildebrandt (2009) and is an implementation of the MiniFab scenario, a simplified semiconductor production line with 5 machines and 2 operators. The machines are arranged in groups as shown in Figure 3, with grouped machines sharing a common job queue. Every job follows a single, fixed, 6-step production process, and each step requires an operator to load the job into a machine before starting and to unload the completed job before it joins the queue of the next step. Jobs arrive dynamically over time according to a Poisson process. We assume that grouped machines (Ma/Mb, Mc/Md) are identical, as well as that all workers are identical and able to operate all machines. Machines can only process one job at a time, and pre-emption is not allowed. We do not consider sequence-dependent set up times for machines, instead assuming that

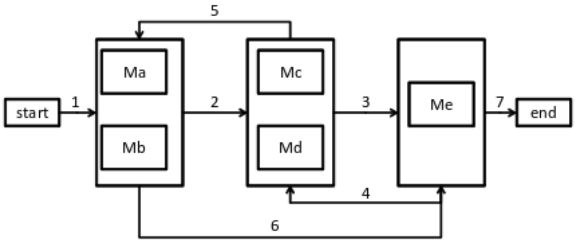

Figure 3: Flow-Shop layout

Table 2: (Un)loading and processing times.

	Machine	Load	Processing	Unload
Step 1	Ma/ Mb	15	55	25
Step 2	Mc/Md	20	25	15
Step 3	Me	15	45	15
Step 4	Mc/Md	20	35	25
Step 5	Ma/Mb	15	65	25
Step 6	Me	10	10	10

the time spent by workers moving between machines is negligible. We also do not consider any batch processing or machine breakdowns in this scenario. The processing times, loading and unloading times for each operation are summarised in Table 2.

As our scenario has only one fixed route and one type of product, we exclude due dates and weights from our simulation. As all finished products are identical, the most urgent order can be fulfilled directly by the next product to leave the production line. As such, the optimal solution to our problem is one that minimizes flow time, which we use as a performance measure for all simulations. This is calculated as the average (mean) difference between starting time and completion time for completed jobs, i.e:

$$MeanFlowTimef = \frac{1}{|J|}\sum_{j\in J}(C_j - r_j)$$

Where J is the set of completed jobs and r_j, C_j the respective arrival and completion times of a job j. As the simulation takes a while to fill with jobs to the desired utilization level, we excluded jobs completed in the first 10% of each simulation run as a warm-up phase.

We use Jasima as a fast simulator of our flow shop scenario to evaluate the fitness of each candidate dispatching rule. Jasima was also used to perform long-term test simulations on final outputs from the GP and to compare against benchmark rules.

4.2 Benchmark rules

We compare our approach to the following six benchmark rules in a long term simulation of the flow shop for a series of different scenarios with different utilization levels and random seeds.

The first four benchmark rules are standard rules from the literature:

- **SPT** - Shortest Processing Time. At each stage the operator selects whichever operation takes least time to complete.
- **FCFS** - First Come First Served. At each stage, the operator selects whichever job has been waiting longest, either in one of the machine queue buffers or on the machine itself.
- **FSFS** - First (in) System First Served. At each stage, the operator selects whichever job has been present in the system longest.
- **WIQ** - Work In Queue. Assigns higher priority to jobs in whichever machine queue contains the most total work-time.

In addition to these four standard benchmarks, we compare our evolved rules against the best performing 2-step heuristics from (Scholz-Reiter, Heger, and Hildebrandt 2009). Because our approach uses a different decision framework (single step rather than 2-step), we had to convert the 2-step approach to fit our framework, while still matching the behaviour of the original rules as closely as possible. In the low (70%) and medium (80%) utilization cases, the best performing heuristic reported by (Scholz-Reiter, Heger, and Hildebrandt 2009) used FSFS for the machine rule and SPT for the operator rule. With this rule

set, the operator would be presented with a choice of several options, namely the oldest job at each idle machine, and would choose whichever has shortest processing time. To replicate this in our framework, we implemented a rule where the operator simply chooses to perform the operation with shortest processing time from the set containing the oldest jobs at each workstation. This should perform similarly or perhaps even better, due to the operator being able to decide between the current oldest jobs in the workstation queues, rather than the oldest job when the workstation became idle and picked a job. Similarly, in the high (90%) utilization case, the best performing heuristic also used FSFS for the machine rule, with WIQ for the operator rule. In our reproduction of this rule, the operator assigns highest priority to the oldest job present in the queue with highest total work-time. We denote our implementations of these rules by **S-R Low/Med** and **S-R High** respectively. Again, we would predict a slightly better performance compared to the original 2-step heuristics as decisions are based on the latest information available when the operator becomes available, rather than the machine making an operator request at the time the machine became available.

5 EMPIRICAL EVALUATION

GP-generated rules were then tested against the benchmark rules in long-term simulations, (replicating a time period of 10 years), for 30 different random seeds and 3 different distributions of job inter-arrival times, as used by Scholz-Reiter, Heger, and Hildebrandt (2009), designed to replicate utilization levels of 70,80 and 90%, for a total of 90 tests per rule.

Each of the evolved heuristics was tested against each of the four benchmark rules as well as S-R Low/Med and S-R High. The results of these tests are summarised in Table 3. We observe that at each of the three utilization levels, the best performing heuristic was the rule evolved by GP for that scenario. Of the simple benchmark rules, FSFS performed best at the low and medium utilization level, with WIQ being by far the best benchmark at the high utilization level. This suggests that in a busy setting it becomes important for the operator to focus on highly queued workstations in order to avoid system overflow. FSFS and SPT, the two best performing benchmark heuristics in the 70 and 80% scenarios are substantially worse than the WIQ and FCFS benchmarks in the 90% scenario. The GP algorithm seems to be able to correctly identify these changing requirements when generating heuristics. In examining the terminal sets of our evolved rules, we noted that the most common terminals present in the rule evolved for the high utilization scenario are WIQ and WINQ, whilst the terminal NRO is absent. In contrast, the rule evolved for the low utilization scenario does contain NRO, but excludes WIQ. These rules are displayed in tree format in Figures 7 and 8. It is interesting to note that the rule evolved for the high utilisation case is relatively robust and also performs reasonably well for lower utilisation levels. The opposite is not true, however, and the rule evolved for the low utilisation case performs rather poorly in the high utilisation scenario. This makes sense, as the high utilisation scenarios are more challenging, and rules that can cope with a more challenging environment would be expected to also perform well in simpler scenarios, but not vice versa. It means, however, that if a shop floor's utilisation level is not precisely known, one should tend to use higher utilisation levels in the simulation during optimisation.

Finally, we note that both S-R low/med and S-R High achieved better performances than their 2-step implementations as reported by Scholz-Reiter, Heger, and Hildebrandt (2009), which we attribute to benefits of using the latest information in the one-step decision making.

Figures 4, 5 and 6 show the pairwise performance of the evolved rules and the best performing benchmark rule at each of the three utilization levels. The performances of the evolved rules are consistently good, beating the best benchmark in 26 of the 30 instances at both the 70% and the 80% utilization level and substantially outperforming the best competitor (S-R High) in all 30 instances at the 90% utilization level. To ensure robustness in these results, we tested for statistical significance, with results shown in Table 4. P-values were calculated using Wilcoxon Rank-Sum, a suitable test as the distribution of a heuristic's mean flowtime for different random seeds is unlikely to be normally distributed, especially if values are close to the optimum. As the table shows, in all three cases, the improvement is statistically significant,

Table 3: Performance of the GP evolved rules and benchmarks at different test utilization levels. Std errors given in brackets and best values in bold.

		Test Utilization		
		70%	80%	90%
	Rule	Mean Flowtime	Mean Flowtime	Mean Flowtime
Training Util.	70%	**656.01** (1.22)	809.56 (4.26)	5322.50 (921.87)
	80%	658.36 (1.23)	**805.11** (3.15)	3522.94 (831.04)
	90%	690.34 (1.32)	854.17 (2.70)	**1304.34** (11.89)
	SPT	695.71 (1.33)	912.16 (4.54)	6709.80 (808.82)
	FCFS	779.33 (2.29)	1085.72 (6.66)	2926.99 (23.70)
	FSFS	663.56 (1.09)	822.61 (2.64)	5241.85 (1015.27)
	WIQ	731.03 (1.55)	958.11 (2.94)	1660.56 (16.14)
	S-R low/med	665.03 (1.07)	827.75 (3.53)	6672.40 (843.20)
	S-R high	731.51 (1.72)	958.52 (3.88)	1650.77 (17.88)

with the GP evolved rule for the high utilization scenario providing approximately 21% improvement in mean flowtime compared to the best alternative.

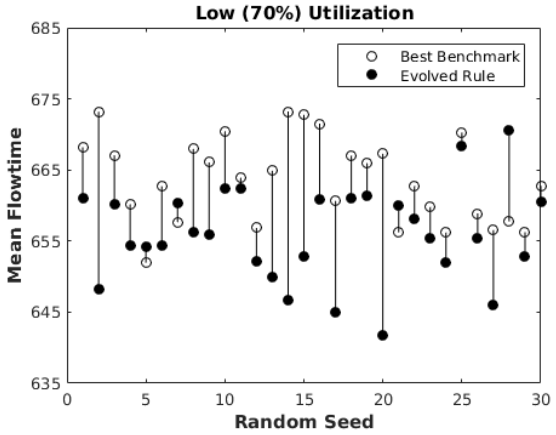

Figure 4: Pairwise comparison of the evolved rule against the best benchmark rule (FSFS) in the low (70%) utilization scenario.

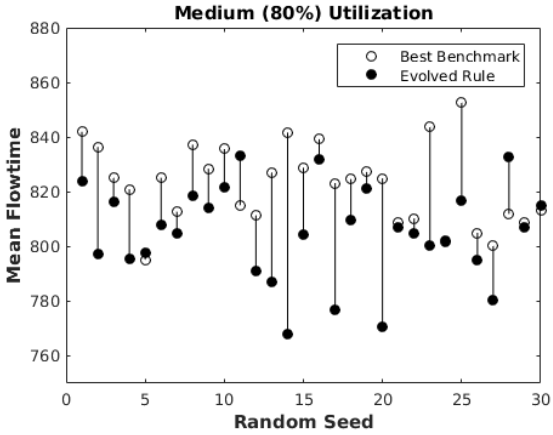

Figure 5: Pairwise comparison of the evolved rule against best benchmark rule (FSFS) in the medium (70%) utilization scenario.

6 CONCLUSION

With the work presented in this paper, we provide a novel framework to handle dual-constrained flow shop scheduling problems where operators are needed to load/unload machines. We show that with this framework, GP is able to automatically generate dispatching rules that performed considerably better than standard benchmarks or rules proposed by Scholz-Reiter, Heger, and Hildebrandt (2009). It should be straightforward to examine the performance of our current GP set-up with the inclusion of additional scenario complications, such as sequence dependent load/unloading times to model, for example, time spent by the operator moving between machines, or to consider objective functions other than flowtime. It would also be interesting to apply a GP-based approach to different dual-constrained manufacturing

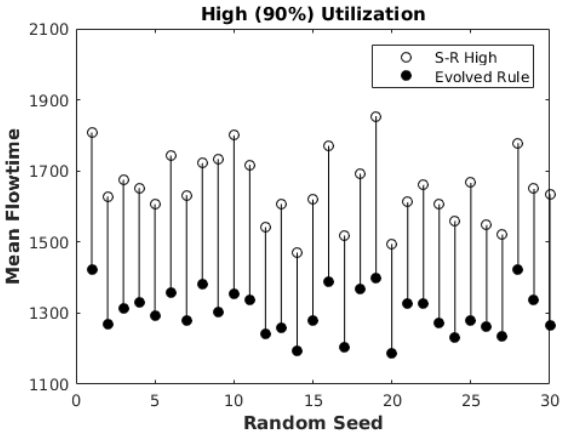

Figure 6: Pairwise comparison of the evolved rule against best benchmark rule (S-R High) for each seed for the high (90%) utilization scenario.

Table 4: Test statistics for the evolved rules. Std error values given in brackets and best values in bold.

Rule	Test Utilization	Mean Flowtime	Best Comparator	P Value	% Improvement
GP Low	70%	**656.01** (4.26)	663.56 (1.09)	4.61×10^{-5}	1.14 (0.24)
GP Med	80%	**805.11** (3.15)	822.61 (2.64)	1.00×10^{-4}	2.13 (0.46)
GP High	90%	**1304.34** (11.89)	1650.77 (17.88)	$< 10^{-10}$	20.99 (0.51)

scenarios, including other features frequently found in modern semiconductor production processes, such as batch processing, or perhaps to cases with more than two resource constraints. Currently little work has been done developing advanced dispatching rules for such problems, so we would anticipate considerable potential for improvement, especially if supported by methods such as hyper-heuristics.

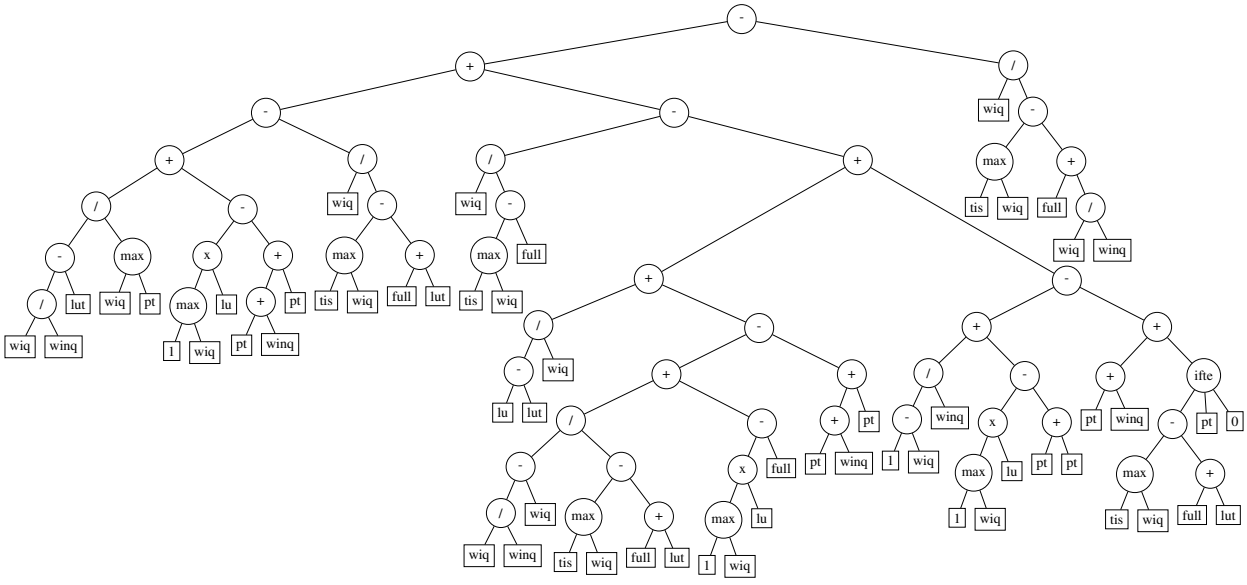

Figure 7: The GP High heuristic displayed in tree form.

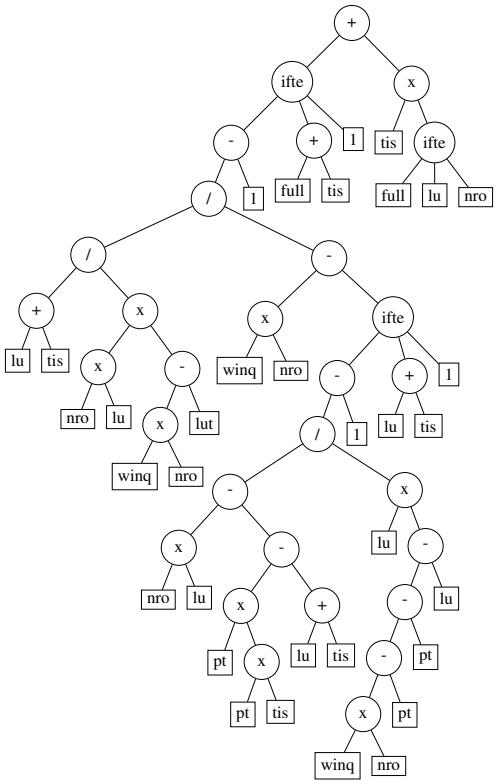

Figure 8: The GP Low heuristic displayed in tree form.

REFERENCES

Branke, J., and T. Hildebrandt. 2015. "Hyper-heuristic evolution of dispatching rules – a comparison of rule representations". *Evolutionary Computation* 23 (2): 249–277.

Branke, J., S. Nguyen, C. Pickardt, and M. Zhang. 2015. "Automated design of production scheduling heuristics: A review". *IEEE Transactions on Evolutionary Computation* 20 (1): 110–124.

Burke, E. K., M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. 2010. "A classification of hyper-heuristics approaches". *Handbook of Metaheuristics* 57:449–468.

Chaudhry, I. A., and P. R. Drake. 2009. "Minimizing total tardiness for the machine scheduling and worker assignment problems in identical parallel machines using genetic algorithms". *International Journal of Advanced Manufacturing Technology* 42 (5-6): 581–594.

ECJ. "A Java-based Evolutionary Compuation Research System". https://cs.gmu.edu/~eclab/projects/ecj/.

ElMaraghy, H., P. Vishvas, and I. B. Abdullah. 2000. "Scheduling of manufacturing systems under dual-resource constraints using genetic algorithms". *Journal of Manufacturing Systems* 19 (3): 186–201.

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

Geiger, C. D., and R. Uzsoy. 2008. "Learning effective dispatching rules for batch processor scheduling". *International Journal of Production Research* 46 (6): 1431–1454.

Geiger, C. D., R. Uzsoy, and H. Aytuğ. 2006. "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach". *Journal of Scheduling* 9 (1): 7–34.

Haupt, R. 1989. "A survey of priority rule-based scheduling". *OR Spektrum* 11 (1): 3–16.

Hildebrandt, T., and J. Branke. 2015. "On using surrogates with genetic programming". *Evolutionary Computation* 23 (3): 343–367.

Hildebrandt, T., J. Heger, and B. Scholz-Reiter. 2010. "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach". In *Genetic and Evolutionary Computation Conference*, 257–264: ACM.

Jasima. "An efficient Java Simulator for Manufacturing and Logistics". http://jasimasolutions.de.

Nguyen, S., M. Zhang, M. Johnston, and K. C. Tan. 2012. "A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems". In *Congress on Computational Intelligence*, 1–8: IEEE.

Nguyen, S., M. Zhang, M. Johnston, and K. C. Tan. 2013. "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem". *IEEE Transactions on Evolutionary Computation* 17 (5): 621–639.

Park, J., S. Nguyen, M. Zhang, and M. Johnson. 2013. "Genetic programming for order acceptance and scheduling". In *Congress on Evolutionary Computation*, 1005–1012: IEEE.

Pickardt, C., J. Branke, T. Hildebrandt, J. Heger, and B. Scholz-Reiter. 2010. "Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness". In *Winter Simulation Conference*, 2504–2515: IEEE.

Pickardt, C., T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter. 2013. "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems". *International Journal of Production Economics* 145 (1): 67–77.

Scholz-Reiter, B., J. Heger, and T. Hildebrandt. 2009. "Analysis and comparison of dispatching rule-based scheduling in dual-resource constrained shop-floor scenarios". In *World Congress on Engineering and Computer Science*, Volume 2, 921–927: IAENG.

Yin, W. J., M. Liu, and C. Wu. 2003. "Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming". In *Congress on Evolutionary Computation*, Volume 2, 1050–1055: IEEE.

AUTHOR BIOGRAPHIES

JÜRGEN BRANKE is Professor of Operational Research and Systems as Warwick Business School, University of Warwick, UK. He has published over 150 articles in international journals and conferences on various topics including multiobjective optimization, handling of uncertainty in optimization, dynamically changing optimization problems, and the design of complex systems, in particular in the area of scheduling and logistics. Prof. Branke is Area Editor of the Journal of Heuristics, Associate Editor of IEEE Transactions on Evolutionary Computation and the Evolutionary Computation Journal. His email address is Juergen.Branke@wbs.ac.uk

MATTHEW J. GROVES is a PhD student in the Mathematics For Real World Systems CDT at the University Of Warwick, UK. His research interests include optimal learning, ranking and selection in noisy environments, and machine learning. His email address is M.J.Groves@warwick.ac.uk.

TORSTEN HILDEBRANDT is co-founder and CEO of jasima solutions UG, offering advanced simulation-based optimization especially in the fields of production and logistics (the company is a recent spin-off of the BIBA - Bremer Institut fr Produktion und Logistik GmbH at the University of Bremen, Germany). His research interests include simulation-based optimization, heuristic optimization algorithms like genetic programming, and planning and control of production and logistic systems. He is the main author of the discrete-event simulation library jasima (JAva SImulator for MAnufacturing and logistics). His email address is hil@jasimasolutions.de.