Towards Improved Dispatching Rules for Complex Shop Floor Scenarios—a Genetic Programming Approach

Torsten Hildebrandt*, Jens Heger, Bernd Scholz-Reiter
Bremen Institute of Production and Logistics – BIBA
at the University of Bremen
Hochschulring 20
28359 Bremen, Germany
{hil,heg,bsr}@biba.uni-bremen.de

ABSTRACT

Developing dispatching rules for manufacturing systems is a tedious process, which is time- and cost-consuming. Since there is no good general rule for different scenarios and objectives automatic rule search mechanism are investigated. In this paper an approach using Genetic Programming (GP) is presented. The priority rules generated by GP are evaluated on dynamic job shop scenarios from literature and compared with manually developed rules yielding very promising results also interesting for Simulation Optimization in general.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*

General Terms

Algorithms, Design, Experimentation

Keywords

Genetic Programming, Job Shop Scheduling, Dispatching Rules, Stochastic System Optimization

1. INTRODUCTION

In todays highly competitive, globalized markets an efficient use of production resources is inevitable for manufacturing enterprises. Therefore especially capital-intensive industries like semi-conductor manufacturing spend considerable effort to optimize their production processes, and as one part of it optimize production scheduling. Scheduling, as many other problems in the manufacturing domain are combinatorial, NP-hard optimization problems. Therefore there is a need for heuristics to solve these problems as optimal solutions can only be obtained for very small problem instances.

One class of scheduling heuristics are dispatching rules which are widely used in industry, especially in complex manufacturing systems like semiconductor manufacturing. Their popularity is derived from the fact that they perform reasonably well in a wide range of environments, and are relatively easy to understand. They also need only minimal computational time, which allows them to be used even in real-time, on-line scheduling environments. Dispatching rules as a special kind of priority rules are applied to assign a job to a machine. This is done each time the machine gets idle and there are jobs waiting. The dispatching rule assigns a priority to each job. This priority can be based on attributes of the job, the machines or the system. The job with the highest priority is chosen to be processed next.

Priority-scheduling rules have been developed and analyzed in the scientific literature for many years, see e.g. [23, 2, 16] . Depending on the manufacturing system and the various objectives (mean flow time, maximum flow time, variance of flow time, proportion of tardy jobs, mean tardiness, maximum tardiness, variance of tardiness, etc.) no single rule has been found, which outperforms all others [25]. For this reason it is a time and cost-consuming process to select the best rule for specific environments and even more so to manually develop good rules. However this is precisely what researchers e.g. in the semi-conductor industry do [7, 8]. This motivates further research especially in the area of automatic adaption and learning of priority rules.

In the evolutionary algorithms (EA) community scheduling received large attention (see e.g. the survey in [15]), however mostly applying it directly to solve instances of scheduling problems. Genetic Programming [21] received only little attention, even though it seems very promising to use it as a hyper-heuristic [5] to automatically develop dispatching rules:

1. the time-consuming GP run can be made off-line, once a good rule was found it can be used as an efficient, on-line/real-time scheduling rule just as any standard rule

2. as an automated approach very little manual work has to be invested, especially to

   (a) adopt to different objective functions

*corresponding author

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

(b) incorporate additional information or assess usefulness of such information for scheduling

3. new priority rules can be easily integrated in existing software for production control and manufacturing simulations.

The work in this paper is a first step to be able to automatically develop dispatching rules for complex manufacturing systems like, e.g. in semi-conductor manufacturing. Therefore we couple a dynamic job shop simulation with a GP component and use the thoroughly researched dynamic job shop scenarios from [25] as an example to develop and test our framework. Primary focus of our work is to find good dispatching rules and compare their performance with manually developed rules for these scenarios and show some of the advantages of our approach, however the paper also addresses the broader topic of coupling a GP search with a stochastic manufacturing simulation to assess individuals' fitness. Therefore the work presented here is also of interest for other uses of simulation optimization.

The paper is organized as follows: in section 2 we give a review of previous work on the use of GP to evolve dispatching rules followed by a description of our framework and the scenarios used in section 3. Section 4 presents our computational experiments and their results. The paper concludes with a short summary and gives directions towards future research.

2. PREVIOUS APPROACHES

One of the first approaches to use Genetic Programming for scheduling problems was conducted by Atlan et al. [1]. They proposed a general system to infer symbolic policy functions for distributed reactive scheduling. As a validating case study of their approach they used the well studied Fisher Thompson [12] static job shop instances and performed very well.

Dimopoulos and Zalzala in [10] investigated the use of GP for the development of priority rules for the one-machine total tardiness problem. The rules they found outperformed standard rules like EDD and SPT as well as the Montagne rule, which was specially developed for the weighted total tardiness problem.

Geiger, Uzsoy and Aytug presented an approach in [13], which is also capable of automatically discovering dispatching rules. It is evaluated in a variety of single machine environments, and discovers rules that are competitive with those in the literature. In a follow-up paper Geiger et al. [14] use their approach to evolve dispatching rules for single machine batch processing.

Jakobović and Budin: [20, 19] proposed a multiple tree adaptive heuristic for job shop scheduling, where a decision tree is used to distinguish between resources based on their load characteristics. The results of their approach exhibited better performance than existing scheduling methods.

Tay and Ho [27] developed dispatching rules for the flexible job-shop problem, where operations can be processed on different machines. This means, that not only the order of operations on a machine (sequencing decision) but also the assignment of operations to machines (routing decision) has to be done. Tay and Ho try to find rules performing well for various objectives by using a linear combination of these objectives in the fitness assessment. For the routing decision they used a fixed "least waiting time assignment" and use GP to evolve sequencing rules. They find rules performing better than standard rules. As our dynamic job shop scenarios can be seen as a special case of a dynamic flexible job shop we also tested their rules on our scenarios. Their performance (ranging from 1.37 to 1.41) was only slightly better than the ERD rule and quite far away from the performance of the SPT rule.

We mainly see three reasons for the poor performance of Tay and Ho's rules in our scenarios: first, they try to find a robust rule working reasonably well not just for the objective of mean flowtime minimization. However even for the single objective of mean flowtime they report their rules to be better than simple rules like EDD or SPT in their scenarios. Second, using the fixed least waiting time assignment to assign operations to machines probably strongly influences the situations faced by the sequencing rules. Third, for fitness evaluations they use sets of problem instances ranging from 10 jobs and 5 machines to 200 jobs and 15 machines. However jobs' release dates are set in an interval of [0,40] or [0,20]. This however results in a sharp load peak (in fact: overloading) at the beginning of the simulation, e.g. in the case of 15 machines all jobs arrive within less than three times the mean operation time. Therefore their rules probably never face typical situations occurring in a long-term simulation of a swung-in manufacturing system and job arrivals modeled as a continuous arrival process, as we use in our work.

To sum up the previous work applying GP to find dispatching rules it can be stated that most work only considered comparatively simple scheduling problems, either single machine or small static scheduling problems. None of the work found in the literature uses a stochastic simulation for performance evaluation of rules on a larger time scale and uses GP on the same scenarios used for the manual creation of dispatching rules.

3. EXPERIMENTAL SETUP

3.1 Problem Description

Our computational experiments use the dynamic job shop-scenarios from [25] and try to find good dispatching rules for these scenarios automatically. In total there are 10 machines on the shop floor, each job entering the system is assigned a random routing, i.e. machine visitation order is random with no machine being revisited. Processing times are drawn from a uniform discrete distribution ranging from 1 to 49. Job arrival is a Poisson process, i.e. inter-arrival times are exponentially distributed. The mean of this distribution is chosen to reach a desired utilization level on all machines.

Following the procedure from [25] we start with an empty shop and simulate the system until we collected data from jobs numbering from 501 to 2500. The shop is further loaded with jobs, until the completion of these 2000 jobs to overcome the problem of censored data [9]. Data on the first 500 jobs is disregarded to focus on the shop's steady state-behavior.

These things being equal we distinguish 4 scenarios varying the utilization level of the system and whether there are missing operations or not:

- *utilization*: is set to 80 % (low/moderate load) and 95 % (highly utilized system)
- *missing/full operations*: in the full setting, each job

entering the system has always 10 operations to complete, i.e. has to visit each machine; with missing operations a job has between 2 and 10 operations (assigned when entering the system using a uniform distribution)

In the experiments using due date information we additionally investigate *flow factor*s of 4.0 and 6.0, resulting in 8 scenarios. A flow factor ff of 4.0 means a job j is assigned its due date D_j as four times its processing time, i.e. if O_j is the set of job j's operations and $p_{i,j}$ the processing time of the ith operation from O_j then $D_j = r_j + \text{ff} \sum_{i=1}^{|O_j|} p_{i,j}$.

Using the standard $\alpha\,|\beta|\,\gamma$-notation from scheduling literature (see, e.g. [24, pp.13] or [3]), where α denotes the problem type, β any specialties involved, and γ is the objective function, the problem as stated above is most similar to $J10\,|r_j(,D_j)|\,FT$, albeit being stochastic and missing operations being different from the standard job shop formulation. The objective function FT denotes mean flow time, i.e. the arithmetic mean of the flow times of jobs 501 to 2500: $FT = \frac{1}{2000}\sum_{j=501}^{2500}(C_j - r_j)$, where C_j is the completion time of a job j and r_j is its release date.

3.2 Benchmark rules

In order to compare our GP-evolved rules with existing dispatching rules, we chose a set of six rules, the first four being standard rules used for decades now. The last two however are rules manually developed by Rajendran and Holthaus and performing very well for the objective of minimizing mean flow time in job shops.

1. *FIFO—First In (queue) First Out:* jobs are processed in the order they entered the queue in front of a machine.

2. *ERD—Earliest Release Date first:* preference is given to the job which entered the system first.

3. *SPT—Shortest Processing Time first:* the job with the shortest processing time of its current operation is processed first. This rule is usually considered a good choice if the objective is to minimize mean flow time.

4. *WINQ—(least) Work In Next Queue first:* jobs are ranked in the order of a (rather worst case) estimation of their waiting time before processing on the *next* machine can start. This estimation includes the time needed by a machine m to finish its current job plus the sum of processing times of all jobs currently waiting in front of m. The job where this sum is least has the highest priority.

5. *PT+WINQ—Processing Time plus WINQ:* this rule uses the sum of WINQ (as just described) and the processing time of a job's current operation. The job where this sum is least gets the highest priority. This rule was proposed by Rajendran and Holthaus in [25] and achieved a very good performance to minimize mean flow time.

6. *2PT+WINQ+NPT* this rule is similar to the just described PT+WINQ-rule but uses twice the Processing Time and adding the processing time of a job's next operation (NPT–Next Processing Time). This rule was manually developed and presented by Rajendran and Holthaus in a follow-up paper to [25]: [17]. In that paper the authors used slightly different scenarios to assess rule-performance, however this rule is also superior in the original scenarios of [25] also used in this paper. This rule is the real benchmark to beat as it is the best rule manually developed for these scenarios.

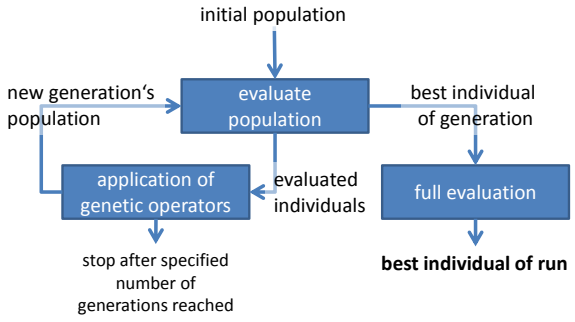

Figure 1: schematic depiction of a single GP run

3.3 Comparing rule performances

The job shop scenarios as described in section 3.1 turn out to have a high variability on one hand due to the high influence of the utilization on mean flow time and results are also rather sensitive to the random numbers used. Therefore multiple independent replications of each scenario are necessary to get reliable estimates of mean flow time values and the comparison has to account for a high variability of flow times between the scenarios. Therefore to compare the performance of two rules A and B we use a performance index perf_B^A over all scenarios from the set of 4 (8) scenarios S. This performance index is defined as:

$$\text{perf}_B^A = \frac{1}{|S|}\sum_{s=1}^{|S|}\frac{FT_{A,s}}{FT_{B,s}} \tag{1}$$

This index is greater than 1 if rule A performs worse than rule B or in the interval $(0,1)$ if A performs better than B. $FT_{A,s}$ is the mean flow time achieved when using rule A in scenario s. As initial experiments showed, the relative performance $\frac{FT_{A,s}}{FT_{B,s}}$ is also less sensitive to the choice of random numbers.

3.4 System Architecture

In our framework to conduct the computational experiments, implemented in Java, we use ECJ [11] to perform the GP-operations. To implement the manufacturing scenarios and assess the performance of dispatching rules we implemented and integrated a simple discrete-event simulation. This simulation is very roughly based on the Java-port of the job shop implementation of the SIMLIB library [22], as described in [18]. Our framework allows the utilization of multi-core processors/computers, which was used in our experiments to run them on an 8-core computer with Intel Xeon 3GHz-CPUs.

Figure 1 shows the general sequence of actions in a single GP run, which follows the standard procedure of most Genetic Algorithms: the starting point is an initial population, which is evaluated. Each evaluation in our case is a simulation run of at least 4 different scenarios (with potentially multiple replications), assessing the performance of an individual, i.e. candidate dispatching rule. As explained in more

Name	Value
population size	1000
generations	20-200 (see text)
crossover proportion	90 %
reproduction proportion	10 %
selection method	tournament selection (size 7)
creation type	ramped half-and-half (min depth 2, max depth 6)
max. depth for crossover	17
function set	$+$, $-$, $\times$, $\div$, max, if3
basic terminal set	PT, NPT, OpsLeft, RemProcTime, TimeInQueue, TimeInSystem, WINQ, 0, 1
due date related terminals	Slack, TimeTillDue, ODD

Table 1: GP parameters used

detail later on we only use a rather simple evaluation here, as a full evaluation would be far too time-consuming. The fitness values of the individuals are then used to apply the standard operators of a GP run to form the population of the next generation. The settings used here are summarized in table 1. This new population needs to get evaluated and the whole procedure is repeated until a certain maximum number of generations were generated and evaluated.

To get a reliable estimate of the performance of a rule requires a simulation with many replications, making it computationally expensive (see section 3.3). Therefore we chose to use a two-step procedure: a "lightweight" evaluation with few replications during the GP run followed by a full evaluation with many replications afterwards. Such a full evaluation is only performed on the best individual of each generation in order to find the best rule of a GP run. This is shown on the right side of figure 1. We chose 200 independent replications to perform a full evaluation and get reliable estimate of a rule's performance, the question of what number of replications to use for the "lightweight" evaluations is investigated in section 4.

Besides choosing the set of terminals and non-terminals and the experiments regarding the best number of replications per fitness evaluation, which vary the maximum number of generations, we did not make any serious attempts to optimize the parameters listed in table 1. These parameter settings are mostly ECJs defaults following recommendations from [21]. We found these settings to work well in our experiments.

3.5 Rule Components

Looking into the literature on dispatching rules and previous uses of GP in similar applications like ours we chose the function set to consist of the basic arithmetic operations augmented by the maximum function and a ternary version of if-then-else (if $a >= 0$ then b else c; a, b, c are sub-expressions evolved with GP). Our basic terminal set consists of the terms listed below. As a dispatching rule gets evaluated each time a machine becomes idle to choose one of its currently waiting jobs, all terms are relative to the job j, which is to be evaluated and assigned a priority.

PT Processing Time of j's current operation

NPT Next Procesing Time, i.e. processing time of j's next operation or 0 if there is no such operation

OpsLeft number of operations left for j

RemProcTime sum of the processing times of all operations left for j

TimeInQueue time j spent in current queue in front of machine m: $\text{TimeInQueue}_{j,m} = t - r_{j,m}$, where $r_{j,m}$ is the point in time j entered the queue

TimeInSystem time spend in system: $\text{TimeInSystem}_j = t - r_j$

WINQ Work In Next Queue as described in section 3.2

constants as constant values we use {0, 1}

For our experiments involving due date information we extended this set of terminals with the following terms:

TimeTillDue $\text{TimeTillDue}_j = D_j - t$

Slack the slack denotes the difference between the time a job gets due and its remaining processing time to finish it $sl_j = \text{TimeTillDue}_j - \text{RemProcTime}_j$. If the slack becomes negative a job can not possibly meet its due date D_j.

ODD Operational Due Date: this is often used as a dispatching rule on its own and assigns each operation i of a job j a due date $D_{i,j}$. There are various ways how to compute these due dates, we chose proportional to an operations processing time. This means if a job has a flow factor ff:

$$D_{i,j} = \begin{cases} r_j + \text{ff} \times p_{1,j} & \text{if } i = 1 \\ D_{i-1,j} + \text{ff} \times p_{i,j} & \text{else} \end{cases}$$

We include this information $(D_{i,j} - t)$ as a terminal in our GP search to have something like intermediate milestones for each operation.

In the formulas above t denotes the current simulation time. Please note that all times are not absolute time values but relative to the current simulation time to ease GP's task of finding rules robust to changes in the simulation length.

3.6 Fitness Function

The fitness f_A of each individual A, i.e. dispatching rule, is evaluated using the formula (lower is better):

$$f_A = \text{perf}^A_{2PT+WINQ+NPT} * \text{fsp}^A_{2PT+WINQ+NPT}$$

Individuals' evaluations are always relative to the fixed baseline rule $2PT + WINQ + NPT$ and consist of the performance index as described in section 3.3 multiplied with a full system penalty fsp^A_B. This factor is usually just 1 but becomes larger than 1 when the simulation has to be terminated early, which can happen if the number of jobs concurrently in the system (work in progress, WIP) gets larger than 500. Such a high WIP is a sure sign of a rule with very bad performance and the system running full. We chose not to assign a fixed very bad fitness to such individuals but still to use its (partial) performance but penalize it with this

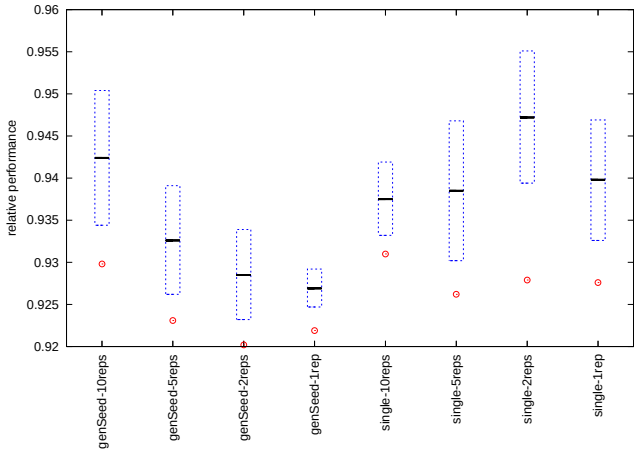

Figure 2: performance of GP rules found using various settings varying the number of replications per evaluation and random seed usage. Experiments 1 to 4 use a new seed in each generation, experiments 5 to 8 use a single seed for each GP run.

factor which gets higher the earlier the WIP threshold was exceeded.

$$\mathrm{fsp}_B^A = \frac{1}{\min\left(0.9, \frac{n_A}{n_B}\right)}$$

In this formula n_A is the number of jobs finished until the simulation was terminated and n_B is the number of jobs the reference rule B needed to complete its simulation run.

4. EXPERIMENTS AND RESULTS

4.1 Evaluation of Stochastic Simulations

In our attempt to find good dispatching rules the most time-critical task is the determination of an individual's fitness. Getting accurate fitness values requires a high number of independent replications, but takes a longer time—using few replications potentially misleads GP's search. To investigate this more closely we ran experiments using different numbers of replications per fitness evaluation. For these experiments we set a maximum of 200000 replications of our 4 scenarios as an upper limit for each GP run. Using a population size of 1000 this limit is reached after a varying number of generations: using just a single replication per evaluation this limit is reached after 200 generations, using 10 replications it is already reached after 20 generations. The underlying question thus is, whether it's more advantageous to spend the 200000 replications in many replications in the hope, GP can cope with the imprecision and, due to the many generations, still arrives at better rules than those using more precise evaluations but fewer generations.

This issue is also related to the question whether using advanced methods from simulation optimization (see e.g., [26]) to use ranking and selection mechanisms like OCBA (Optimal Computing Budget Allocation, [6]) in a GP/GA run can be beneficial for our work.

Another factor to investigate in these experiments is the question of proper random number use. In all cases we use common random numbers [22, pp. 578] to evaluate all individuals of a certain generation, that is, all individuals of a generation face exactly the same shop floor situation. The question however is if it's better to use a new random seed for each generation or is it better to use a single seed for a complete GP run, so whether a single seed causes some kind of overfitting to the special conditions of a single run or if a single seed is actually beneficial because otherwise GP's search gets less effective as it is has to cope with a slightly different situation each generation.

Figure 2 shows some results of these experiments. Experiments 1 to 4 use a new seed in each generation, experiments 5 to 8 use a single seed for each GP run. The number of seeds is varied: 10, 5, 2, 1, that is experiment 1 uses 10 evaluations and a new seed per evaluation, and experiment 8 uses a single seed per run and just a single replication per evaluation. The boxes show the average solution quality after the last generation, i.e. 200000 replications run, averaged over 10 runs. The black line in the box is the mean value, the blue box denotes its 95 % confidence interval. The red points below the boxes show the best rule found in any of the runs and generations (as already stated before we save and fully evaluate the best individual of each generation).

Looking at experiments 1 to 4, when using a new seed for each generation, there is a trend towards just a single replication per evaluation. Although only the confidence intervals of 10 and 1 replication(s) don't overlap using the replications for as many generations as possible seems to be the best choice to get a good rule. The situation is less clear when looking at the results of experiments 5 to 8 using only a single seed. Figure 3 shows them in more detail, plotting the development of the best rule found after a certain number of generations. Additionally the graph contains the same curve for the best setting for a new seed each generation, i.e. using just 1 replication.

As the graphs show in the cases of 1 and 2 replications (light blue and purple line) solution quality stagnates quite early. This means in these cases overfitting of the rules to the specific situation created by the single random number seed occurs, even if a rule's performance further increases for this specific situation encountered in a certain GP run, when fully evaluating them their performance for the general problem does not get better any more. Using 5 or 10 replications per evaluation however seems to make little difference compared to their counterparts using a new seed each generation. In these cases the diversity in an evaluation seems to be high enough to be representative for the problem in general.

As the result of these experiments we chose a single replication per evaluation and a high number of generations using a new random number seed for each generation as parameters for our subsequent experiments. It also means the use of ranking and selection procedures like [26] mentioned before does not make sense to speed up GP's evaluations in our case, the efficiency of just a single replication per evaluation can hardly be improved upon.

4.2 Additional Domain Information

When using an automated approach like ours it is very easy to include additional information into dispatching rules. During the GP run the genetic algorithm automatically finds a good way to integrate such information in a rule—or finds rules without it if rule performance does not benefit from

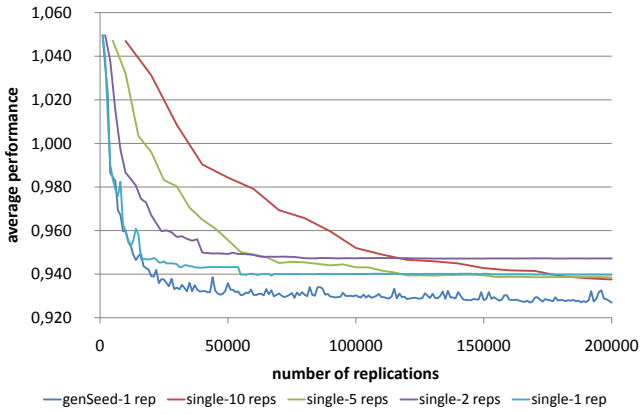

Figure 3: average performance of best rules after a certain number of replications over 10 runs for different settings of replications per evaluation.

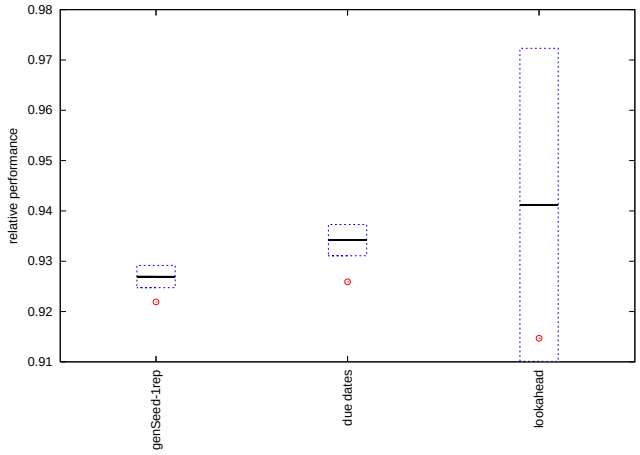

Figure 4: Results of experiments to include additional information.

them. We illustrate this with two of our experiments, one integrating due date-related information into the search and in section 4.2.2 we use the information on jobs arriving in the near future in our rules.

4.2.1 *Due Date Information*

Even though at first it seems not clear why a rule optimized for best mean flow time performance should contain information on due dates, the rational behind these experiments however was the fact that in the Holthaus/Rajendran-papers [25, 17] sometimes rules including due date information achieve a very good mean flow time. Amending our basic terminal set with the due date-related terminals as listed in table 1 we checked, whether this information can also be beneficial for GP runs. The results are shown in figure 4, experiment "due dates". This diagram is similar to figure 2, i.e. the blue dashed boxes show the confidence interval around the black vertical line showing the mean performance reached at the end of 10 GP runs. The red dots illustrate the performance of the best rule found in the course of these 10 runs.

Comparing the results with the best parameter combination "genSeed-1rep" identified before and also included in figure 4 for reference, it is clear that including due date information is not beneficial, mean performance at the end is clearly worse (confidence intervals don't overlap) and also the best individual found is only in the range of mean performance of "genSeed-1rep".

4.2.2 *Lookahead*

The information on jobs arriving in the near future can improve scheduling decisions. Therefore we implemented an efficient way to make this information available to dispatching rules in our job shop information. Whenever a job starts processing on a machine m the machine $m+1$ this job is going to visit next is notified of its arrival at a certain point in time in the near future and enqueued in its queue, however marked as a job not currently present in the queue. Assigning the highest priority to one of these future jobs, a dispatching rule now has the possibility to keep a machine idle. This allows them to create a broader class of schedules than the non-delay schedules that are able in the experiments presented before (for the classification of schedules in non-delay, active, semi-active see, e.g. [24, pp. 21]). At least in theory this means our dispatching rules are now able to find optimal schedules for more scenarios.

The information on future jobs is in our experiment reflected in two terminals: the TimeInQueue-term now becomes negative for these future jobs and in addition to the basic terminal set as listed in table 1 we include WINQ2 as an additional terminal. The only difference between WINQ and WINQ2 is WINQ2 including the processing times of future jobs to arrive on the next machine, whereas WINQ does not.

Looking at the results ("lookahead" in figure 4) the mean performance at the end of a GP run is rather bad with about 94 %. The variance of these values however is quite high as is reflected in the broad confidence interval. This means, in some runs only rather bad solutions can be found whereas also very good rules are possible. This is also reflected in the best rule found: with a performance of 0.9147 it is on average about 0.5 % better than the best rule found using just the basic terminal set and no lookahead: "genSeed-1rep" with a performance of 0.9202.

One reason for this high variance is a more difficult search caused by the additional degree of freedom for dispatching rules to keep a machine idle. If not used properly, especially in the scenarios with 95 % utilization, you very easily end up with the system running full.

4.3 Best Rules Found

To summarize our experiments table 2 lists the results of our benchmark rules and the best rules found in the experiments described before. It shows for each of the 4 scenarios the mean flow times achieved. The values in brackets are the standard error of these means over the 200 independent replications we used to obtain them. Column 6 contains the mean flow time averaged over all scenarios, and the last column contains the performance relative to 2PT+WINQ+NPT. The 3 rules highlighted are our reference rule 2PT+WINQ+NPT, the best rule found using the basic terminal set and no lookahead "genSeed-1rep" and the rule found with the best overall performance "lookahead" utilizing information on future jobs.

One drawback of the best rules found however is their

	full job shop		missing operations			
Name	80%	95%	80%	95%	Mean	Perf
FIFO	822.5 (4.6)	2292.4 (36.7)	512.2 (3.6)	1440.4 (23.6)	1266.9	**1.6297**
ERD	791.9 (3.9)	1878.9 (24.0)	496.5 (3.1)	1261.7 (18.1)	1107.3	**1.4244**
SPT	619.3 (2.4)	1377.3 (22.1)	387.4 (2.0)	935.5 (17.1)	829.9	**1.0675**
WINQ	684.0 (2.8)	1554.4 (23.8)	430.5 (2.4)	999.5 (16.0)	917.1	**1.1797**
PT+WINQ	619.4 (2.4)	1362.1 (20.9)	386.9 (2.1)	888.1 (14.9)	814.1	**1.0473**
2PT+WINQ+NPT	**611.5 (2.2)**	**1273.1 (18.6)**	**383.9 (1.9)**	**841.0 (13.6)**	**777.4**	**1.0000**
genSeed-1rep	588.5 (2.0)	1152.1 (16.0)	367.9 (1.7)	758.2 (11.6)	716.7	**0.9219**
genSeed-2reps	**589.6 (2.0)**	**1147.9 (15.7)**	**368.0 (1.7)**	**755.9 (11.5)**	**715.4**	**0.9202**
genSeed-5reps	588.7 (2.0)	1154.1 (15.8)	367.5 (1.7)	760.1 (11.6)	717.6	**0.9231**
genSeed-10reps	589.0 (2.0)	1164.8 (16.8)	367.9 (1.7)	769.3 (12.3)	722.8	**0.9298**
single-1rep	589.7 (2.0)	1160.9 (16.7)	367.8 (1.7)	766.1 (12.3)	721.1	**0.9276**
single-2reps	590.0 (2.0)	1161.2 (16.7)	368.1 (1.7)	766.2 (12.2)	721.4	**0.9279**
single-5reps	589.6 (2.0)	1157.9 (16.2)	368.9 (1.7)	763.7 (11.9)	720.0	**0.9262**
single-10reps	591.1 (2.0)	1166.4 (16.7)	368.7 (1.7)	768.8 (12.3)	723.8	**0.9310**
due_dates	589.3 (2.0)	1157.5 (16.1)	367.9 (1.7)	764.5 (12.0)	719.8	**0.9259**
lookahead	**581.0 (1.9)**	**1142.9 (16.5)**	**363.6 (1.7)**	**756.6 (12.2)**	**711.0**	**0.9147**

Table 2: Flow times in simulation scenarios for various rules and mean performance relative to 2PT+WINQ+NPT, lower values are better. The numbers in brackets give the standard error after 200 replications.

size—good rules tend to get rather complex in the number of terminals and non-terminals. Therefore we can only give one of the rules here, "genSeed-10rep". It computes a priority value Z (the higher the better) using the formula shown below. In this formula abbreviations were used for the terminals: $p_{i,j}$ is PT, $p_{i+1,j}$ is NPT, n_j^L is OpsLeft, w is WINQ, tiq is TimeInQueue, and tis is TimeInSystem:

$$Z = -p_{i,j}\left[\left(p_{i+1,j} - \frac{p_{i+1,j}}{p_{i,j}}\right) w + \left[\max\left(p_{i,j}, n_j^L - \text{tiq}\right) \times \left(\max\left(p_{i,j} - p_{i+1,j}, \frac{p_{i,j}}{p_{i,j} + \text{tis}}\right) + 1\right) + 1\right]\right]$$

5. RULE ROBUSTNESS

As shown in the previous section, GP was able to find very good dispatching rules for our scenarios. To check the generalization abilities of the rules, we modified our experimental setting in two ways:

- *Number of machines*: besides the maximum number of 10 machines we also use 50 machines. This means the number of operations per job increases to 50, or lies in the interval [2,50] for the scenarios with missing operations.
- *Distributions*: instead of an exponential distribution for inter-arrival times and a discrete uniform distribution for processing times we use a uniform distribution, and a log-normal distribution respectively. Distribution parameters were set so mean inter-arrival times as well as mean and variance of the processing times remain the same.

These two settings for the number of machines and whether to use the original or changed distributions creates 4 different settings, one of which being the original setting used in the experiments presented before.

The results of these experiments are shown in Table 3. Irrespectively of the distributions and number of machines the rules evolved by GP can maintain their good performance. Increasing the number of machines results in a decrease of performance compared to using 10 machines, although results are still better than our benchmark rules.

	orig. distr.		altered distr.	
Name	m=10	m=50	m=10	m=50
SPT	1.0675	1.0726	0.9571	1.0016
genSeed-1rep	0.9219	0.9452	0.9091	0.9408
genSeed-2reps	0.9202	0.9347	0.9050	0.9339
genSeed-5reps	0.9231	0.9335	0.9120	0.9333
genSeed-10reps	0.9298	0.9473	0.9104	0.9367
single-1rep	0.9276	0.9442	0.9094	0.9360
single-2reps	0.9279	0.9555	0.9073	0.9463
single-5reps	0.9262	0.9372	0.9119	0.9377
single-10reps	0.9310	0.9780	0.9097	0.9531
due_dates	0.9259	0.9594	0.9490	0.9723
lookahead	0.9148	0.9309	0.8936	0.9192

Table 3: Performance of selected rules relative to 2PT+WINQ+NPT in robustness experiments.

One other interesting thing happens: changing distributions changes the performance of SPT. In the case of 10 machines it is now clearly better than the reference rule, with 50 machines they achieve about the same performance.

6. CONCLUSION AND OUTLOOK

With the work presented in this paper we were able to find much better dispatching rules for the dynamic job shop scenarios of [25]. Using their own scenarios we were able to beat their best manually developed rule presented in [17], a follow-up paper of [25], by a large extend. Using their 2PT+WINQ+NPT rule brings an improvement of 6.3 % over SPT on average. In our work we were able to improve over their rule by another 8.5 %, if compared with SPT this is an improvement of 14.3 %.

Robustness experiments indicate GP was able to find rules generalizing well to changing conditions in the form of different distributions and/or increased number of machines.

Besides the success of our GP approach in these particular job shop scenarios our work addresses the broader topic of coupling a GP search with a rather time-consuming stochastic manufacturing simulation to evaluate individuals. We found in our case using just a single replication per evaluation and a new random number seed for each generation to yield the best results.

Extending our work to also address other objective functions besides mean flow time, such as mean tardiness, number of tardy jobs, or combinations thereof, is very straightforward. A more challenging task is to extend the manufacturing system simulation of our framework to address things like sequence-dependent setups, parallel batching machines etc. and make good scheduling decisions in such environments. As we could show even in the case of the thoroughly researched dynamic job shop scenarios we used in this work there was still room for larger improvements. In the more complex settings just outlined substantially less work was done, so we expect even more potential in these cases.

Even though our experiments showed (see section 4.1) that the use of methods like OCBA does not make sense for our work to speed up GP's fitness evaluations, there is still potential to use such methods to make full evaluations more efficient using an approach similar to [4].

It would also be interesting to see how good our rules are compared to more complex heuristics that are able to

solve a problem like ours, e.g. a Genetic Algorithm with a sliding time window, or something similar. Such a lower bound on what is possible in our scenario would be very helpful to assess the effectiveness of the GP search but also to assess how good scheduling with local dispatching rules can get. We can clearly show our rules to be much better than standard and existing rules but unfortunately we don't know for sure what is possible in our scenarios.

7. ACKNOWLEDGMENTS

The authors would like to thank Jürgen Branke and Christoph Pickardt, University of Warwick, for many fruitful discussions helping to improve this paper substantially.

This research is funded by the German Research Foundation (DFG) under grant SCHO 540/17-1.

8. REFERENCES

[1] L. Atlan, J. Bonnet, and M. Naillon. Learning distributed reactive strategies by genetic programming for the general job shop problem. May 1994.

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

[3] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer, Berlin et al., 2nd edition, 2001.

[4] J. Boesel, B. L. Nelson, and S.-H. Kim. Using ranking and selection to "clean up" after simulation optimization. *Operations Research*, 51(5):814–825, 2003.

[5] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward. A classification of hyper-heuristics approaches. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Meta-heuristics*. Springer, 2nd edition. Forthcoming.

[6] C. H. Chen, D. He, M. Fu, and L. H. Lee. Efficient simulation budget allocation for selecting an optimal subset. *Informs Journal on Computing*, 20(4):579–595, 2008.

[7] C.-C. Chern and Y.-L. Liu. Family-based scheduling rules of a sequence-dependent wafer fabrication system. *IEEE Transactions on Semiconductor Manufacturing*, 16(1):15–25, 2003.

[8] T. C. Chiang, Y. S. Shen, and L. C. Fu. A new paradigm for rule-based scheduling in the wafer probe centre. *International Journal of Production Research*, 46(15):4111–4133, 2008.

[9] R. W. Conway. Priority dispatching and job lateness in a job shop. *Journal of Industrial Engineering*, 16:228–237, 1965.

[10] C. Dimopoulos and A. M. S. Zalzala. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, 32(6):489–498, 2001.

[11] ECJ. A Java-based Evolutionary Computation Research System, project homepage, 2009. http://cs.gmu.edu/~eclab/projects/ecj/, last accessed 15th January 2010.

[12] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice-Hall, 1963.

[13] C. Geiger, R. Uzsoy, and H. Aytug. Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. *Journal of Scheduling*, 9(1):7–34, 2006.

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

[15] E. Hart, P. Ross, and D. Corne. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines*, 6(2):191–220, 2005.

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

[17] O. Holthaus and C. Rajendran. Efficient jobshop dispatching rules: further developments. *Production Planning & Control*, 11(2):171–178, 2000.

[18] B. J. Huffman. An object-oriented version of SIMLIB (a simple simulation package). *INFORMS Transactions on Education*, 2(1):1–15, 2001.

[19] D. Jakobović and L. Budin. Dynamic scheduling with genetic programming. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 73–84, Budapest, Hungary, 10 - 12 Apr. 2006. Springer.

[20] D. Jakobović, L. Jelenković, and L. Budin. Genetic programming heuristics for multiple machine scheduling. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 321–330, Valencia, Spain, 11 - 13 Apr. 2007. Springer.

[21] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[22] A. M. Law. *Simulation modeling and analysis*. McGraw-Hill, Boston, USA et al., 4. edition, 2007.

[23] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

[24] M. Pinedo. *Scheduling—Theory, Algorithms and Systems*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2nd edition, 2002.

[25] C. Rajendran and O. Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, 116(1):156–170, July 1999.

[26] C. Schmidt, J. Branke, and S. E. Chick. Integrating techniques from statistical ranking into evolutionary algorithms. *Springer Berlin / Heidelberg*, 3907:752–763, 2006.

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