

# Evaluating Online Scheduling Techniques in Uncertain Environments

Michael Thomas, Helena Szczerbicka

FG Simulation & Modellierung, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany  
 {mit, hsz}@sim.uni-hannover.de

---

Online scheduling (also known as dynamic scheduling) refers to the real-time coordination of operation processing while new operations are continuously arriving. The objective of this paper is the study of different online scheduling techniques in a dynamic, stochastic job shop scheduling environment. We present a quantitative study of the quality of dispatching and optimizing scheduling techniques under varying uncertainty and utilization. Therein, we also consider a regret-based algorithm adopted from Bent and Van Hentenryck [1] in a setting with stochastic processing times and analyze its performance. To date, results in such a dynamic setting are lacking, while it has been shown to outperform other heuristic optimization methods in online packet scheduling with processing times known a priori.

*Keywords:* Online Scheduling, Job-Shop Scheduling, Heuristic Search.

---

## 1 Introduction

In recent years, automated scheduling techniques have received increasing attention. Such techniques have great impact in reducing the operational cost and in increasing the autonomy of complex manufacturing systems. Contrary to static job shop scheduling problems, the data to be processed is not known in advance but rather revealed over time. The task of the scheduling system is to decide which request has to be served next while minimizing a given objective function, e.g. the maximum or weighted average lateness.

While the notion of dynamic scheduling is frequently used in computer architectures as the technique for speeding up execution, this paper refers to dynamic scheduling (also called online scheduling) in a dynamic, stochastic job shop setting. That is a job shop with stochastic operation processing times and jobs, composed of a sequence of (atomic) operations, that arrive over the course of time. We allow precedence delays, transition times between different operations on a resource and—unlike most publications on online scheduling—an operation to require multiple resources to be processed. However, in operation processing resource-sharing and preemption are not allowed.

In this setting, we compare the quality of schedules obtained from (a) a dispatch rule, i.e. sequentialization by rules, and (b) sequentialization by optimization algorithms, with respect to the variances of the random variates in the problem instance. Furthermore, we analyze the performance of the Regret approach proposed by Bent et al. [1, 2] since it lets open how and whether sampling of future events can still improve a combinatorial optimization algorithm in a scenario with stochastic timings.

The remainder of the paper is structured as follows. In Section 2, related work on online scheduling and the dynamic aspects of scheduling problems is briefly discussed. Section 3 introduces the dynamic job shop scheduling problem and our framework. In Section 4, the scheduling algorithms are described. Section 5 contains experiments and results, and Section 6 concludes the paper.

## 2 Related Work

For a long time, research mostly concentrated on scheduling as a static problem. In recent years, this perspective shifted towards the dynamic aspects and appropriate scheduling techniques. These dynamic scheduling techniques can be roughly distinguished into the following three branches that we will summarize briefly.

The first approach to dynamic scheduling is dispatching with rules. This approach is very common and well suited for systems where short response times are required and the future is uncertain, such as operation systems or network routers. A variety of rules has been developed (for a survey, see e.g. [9]). However, rule performance depends on the problem instance, thus the challenge is to determine which rules perform best [7]. This issue has been addressed with dynamic rule selection using iterative learning and heuristics [6, 12]. Nevertheless, these schedulers generally lack the ability to plan into the future what renders them suboptimal in the presence of stochastic prediction models.

The second approach to dynamic scheduling derives from the static methods. It continuously recomputes the schedule via combinatorial optimization to reflect changes in the environment, like newly arriving jobs, etc. That is, periodically or upon significant events, a snapshot of the system is taken and processed by algorithms for static scheduling, see e.g. [5, 3]. This approach has been extended by Sun et al. [10] or Smith [11] to distinguish severity among the changes: scheduling is divided into a planning phase, that recomputes the whole schedule, and an execution phase in which the current schedule is repaired by a light-weight scheduler. Approaches of the second kind are well suited in predictable environments with a time-scale large enough for schedule computation.

The third approach to dynamic scheduling is stochastic optimization, in which schedule (re-) computation is done with respect to a stochastic model of the future. While this approach seems similar to the second one, it explicitly integrates future events and adjusts the schedule accordingly. Therefore most algorithms sample future events from a stochastic (or empirical) model for each possible scheduling decision. Bent and Van Hentenryck propose an algorithm that enables the evaluation of every decision on all samples, that is without distributing the samples among them [1]. In [4], Chang et al. formulate a partially observable Markov decision process for simplified scheduling problems (in particular, the authors assume the absence of precedence constraints) using a hidden Markov model for the arrival process.

Still, an issue pointed out by Montana in [8] remains unsolved. In the presence of limited computational power and varying processing times, is it still reasonable to compute an optimized schedule? Or does, under these conditions, dispatching perform equally well with less costs? Hence, in the following we will analyze under which conditions combinatorial optimization (with and without sampling of the future) reasonably outperforms dispatching rules and how sampling may improve the optimization results.

## 3 The Job Shop Framework

For the sake of clarity, we first formally define a dynamic stochastic job shop problem and then describe our framework.

### 3.1 Formal Definition

An instance of the dynamic stochastic scheduling problem, in the following also referred to as *scenario*, consists of basically three types of entities: *resources*, *jobs* and *operations*. Jobs are temporal entities that continuously arrive into the system and specify a sequence of operations. These operations have to be processed sequentially in the given order to complete the job. Resources

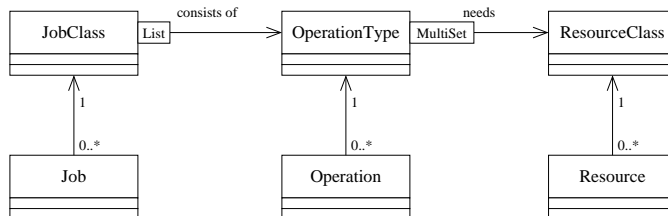


Figure 1: Structure diagram of a scenario. Arrows indicate associations.

are permanent entities needed for operation processing. Each resource can process at most one operation at a time. There are resource-dependent transition times between different operations; preemption is not allowed. Resources are partitioned into *resource classes*, jobs into *job classes* and operations into *operation types*, so that all entities in each class and type share the same properties:

**Resource Classes.** A resource class specifies transition (or setup) times between operation types and the number of resources in that class.

**Operation Types.** An operation type specifies the number of resources per resource class needed for its execution and a probability distribution of its execution times.

**Job Classes.** A job class specifies the sequence of operation types to be processed in order to complete the job, a sequence of precedence delays, a fixed weight and probability distributions for the interarrival and execution time (the time between a job's release and its deadline).

Figure 1 shows a diagram of the resulting structure. To define the dynamics of a scenario, consider a finite time horizon  $H = [0, h]$ ,  $h \in \mathbb{R}$ . A configuration of a dynamic scheduling instance at time  $t$ ,  $0 \leq t \leq h$ , can be described as a quadruple  $I(t) = (R, J, O(t), f(t))$ , where  $R = (R_1, \dots, R_n)$  denotes a vector of available resources per resource class,  $J = (J_k)_{1 \leq k \leq m}$  is the vector of job arrival processes with one component for each class,  $O(t)$  is the set of arrived but unserved operations and  $f: R \rightarrow \mathbb{R}_{\geq 0}$  is a mapping that associates with each resource  $r \in R$  its latest release time  $\leq t$  (i. e. the latest point in time  $\leq t$  that an operation using resource  $r$  completed). Further on, we have to distinguish operations in  $O(t)$  into schedulable and unschedulable operations, since arriving jobs introduce a sequence operations whose members have to be processed sequentially with respect to the given precedence delays. We will refer to the set of schedulable operations at time  $t$  as  $O_{\text{sched}}(t)$  and define the release time of operation  $o$  as  $a_o = \min\{t \mid o \in O_{\text{sched}}(t)\}$ ; the associated job will be denoted  $j_o$ . We will also refer to  $J$  as the set of all jobs arriving in  $[0, H]$  and  $O$  as the set of all operations, respectively.

A solution to an online scheduling problem is a mapping  $s: R \times H \rightarrow \bigcup_{t \in H} O_{\text{sched}}(t)$ , called *schedule*, such that  $s(t) = o$  if operation  $o \in O(t)$  is scheduled at time  $t$ , and  $s(t) = \text{undefined}$ , otherwise.

The goal of an online scheduler is to compute  $s(t)$  with the knowledge of  $I(t)$  while minimizing a given objective function. In this paper, we fix this function to the weighted average lateness  $\frac{1}{|J|} \sum_{j \in J} (c_j - d_j)$ , where  $c_j$  denote the completion time of the last operation and  $d_j$  the due date of job  $j$  (other examples for an objective function are e. g. the maximum lateness  $\max_{j \in J} \{c_j - d_j\}$ ).

### 3.2 Framework

We set up a modular framework, to be able to evaluate all online scheduling techniques under exactly the same conditions. Basically, this framework is composed of four components: the simulator driving the discrete event simulation, a scenario describing an instance of a dynamic, stochastic

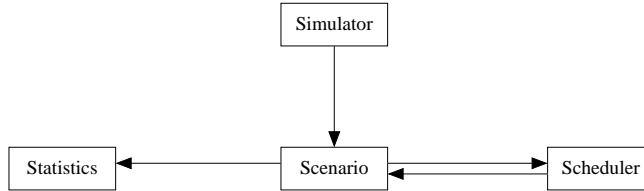


Figure 2: The block diagram of the framework. Arrows indicate control flow during execution.

<pre> 1 <b>function</b> CHOOSEOPERATION(): 2   <math>RC \leftarrow \text{AVAILABLERESOURCES}()</math> 3   <math>opList \leftarrow \text{COMPUTE PRIORITYLIST}(O_{\text{sched}}(t))</math> 4   <b>for</b> (<math>i \leftarrow 1</math> <b>to</b> <math> O_{\text{sched}}(t) </math>) <b>do</b> 5     <b>if</b> (<math>\text{NEEDEDRESOURCES}(opList[i]) \leq RC</math>) <b>then</b> 6       <math>\text{SCHEDULE}(opList[i])</math> 7       <math>RC \leftarrow RC - \text{RESOURCECLASSES}(opList[i])</math> 8     <b>end if</b> 9   <b>end do</b> 10 11 12 </pre>	<pre> 1 <b>function</b> COMPUTEPRIORITYLIST(<math>O_{\text{sched}}(t)</math>): 2   <b>for</b> (<math>o \in O(t)</math>) <b>do</b> 3     <math>r(o) \leftarrow 0</math> 4   <b>end do</b> 5   <b>for</b> (<math>i \leftarrow 1</math> <b>to</b> <math>k</math>) <b>do</b> 6     <math>F \leftarrow \text{SAMPLEFUTUREEVENTS}()</math> 7     <math>S \leftarrow \text{COMPUTESCHEDULEGA}(O(t) \cup F)</math> 8     <b>for</b> (<math>o \in O(t)</math>) 9       <math>r(o) \leftarrow r(o) + \text{EVAL}(s) + \text{REGRET}(s, o)</math> 10    <b>end do</b> 11  <b>end do</b> 12  <b>return</b> <math>\text{SORT}(O_{\text{sched}}(t), r)</math> </pre>
--	--

Figure 3: The generic scheduling algorithm.

Figure 4: The Regret priority computation.

job shop, a scheduler reacting to the events in the scenario and a data acquisition module tracking the desired statistics. Figure 2 shows the control-flow in the framework. In a preprocessing step, all stochastic times are drawn from their respective distributions to ensure an equal amount of variance in the each simulation. In particular, this enables accounting differences directly to the scheduling technique. Then, for each scheduler, the simulator processes the events generated in the preprocessing step in increasing timestamp order, while events, on their part, trigger the scheduler and log the statistics afterwards.

We have validated the framework using the scenarios and dispatch scheduler from [8] and obtained the same results within a range of 5% that we credit to missing details in the framework description.

## 4 Scheduling Algorithms

This section introduces the online scheduling algorithms considered in this paper. The algorithms were selected as representatives for the different scheduling techniques mentioned in Section 2, and chosen for two reasons. First, they provide reasonable performance compared to other schedulers of the same type. And second, they are relatively straightforward implementations and have already been treated in several papers.

Upon event occurrences, all scheduling algorithms are triggered by a call to the function `CHOOSEOPERATION()` shown in Figure 3. There, the function `AVAILABLERESOURCES()` returns a vector containing the number of free resources per class, whereas `NEEDEDRESOURCES(o)` returns the number of resources per class required to execute operation  $o$ . The function `SCHEDULE(o)` schedules operation  $o$  and, at last, the function `COMPUTE PRIORITYLIST(O)` returns a list of operations in  $O$ , ordered according to the sequence in which they shall be scheduled. This is the only function that the following schedulers differ in.

#### 4.1 Dispatch-Rule Scheduler

Dispatch-rule schedulers do not look ahead in terms of planning schedules. Their general approach is to rank each operation according to a given dispatch rule and execute the operation that maximizes (or minimizes) that score as soon as all required resources can be allocated.

In this paper, we consider two schedulers of this kind: one implementing a *first-come-first-served* (FCFS) strategy, the other one implementing an modified *least-slack* (LS) strategy. For the FCFS scheduler, `COMPUTEPRIORITYLIST( $O_{\text{sched}}(t)$ )` returns the operations ordered by ascending release times  $a_o$ ,  $o \in O_{\text{sched}}(t)$ ; for the LS scheduler, the list is ordered ascending according to

$$\pi(o) = \frac{s_{\text{initial}}(j_o)}{s_{\text{now}}(j_o)} \cdot w_{j_o} \cdot (P_{\text{assign}}(o) - P_{\text{wait}}(o)),$$

where  $o$  is an operation,  $w_j$  the weight of job  $j$ ,  $s_{\text{initial}}(j)$  its initial and  $s_{\text{now}}(j)$  its current slack time<sup>1</sup>,  $P_{\text{assign}}(o)$  the probability that  $j_o$  fails to complete on time if  $o$  is scheduled now and  $P_{\text{wait}}(o)$  the probability that  $j_o$  fails to complete on time if  $o$  is not scheduled now. That is,  $\pi(o)$  equals the ratio of  $j_o$ 's initial slack time to its current slack time multiplied with  $j_o$ 's weight and the probability loss of not being late. The estimation of  $P_{\text{assign}}(o)$  and  $P_{\text{wait}}(o)$  has been done as in [8].

#### 4.2 Optimizing Scheduler

Optimizing schedulers treat the dynamic job shop as a static problem. They compute a schedule for all arrived but unserved operations using combinatorial optimization algorithms without consideration of future events and times. To this end, the unknown stochastic processing times are substituted with their mean values.

The optimization algorithm applied is the genetic algorithm described in [3] using the permutation with repetition encoding and the hybrid Giffler-Thompson algorithm with  $\delta = 0.5$  for the decoding step. The population size is kept constant at  $\min\{|T(t)|, 100\}$ . We use elitism for 20% of the population and generalized order crossover for the remaining 80%—this crossover operator tends to respect the relative order of operations. Mutation is applied with a probability of 1%. The optimization criteria employed are estimated maximum and weighted average lateness. The genetic algorithm is initialized with chromosomes from the dispatch schedulers above, iterated for 30 generations, and finally, returns the operations in  $O_{\text{sched}}(t)$  ordered according to best chromosome found.

#### 4.3 Stochastic Optimizing Scheduler

A stochastic optimizing scheduler works similar to an optimizing scheduler but additionally incorporates sampled future job arrivals and processing times from a stochastic (or empirical) model of the job shop. This way, the schedules are expected to be more robust to strongly deviating processing times.

We have developed an algorithm which is an adoption of the Regret approach in [1]. The Regret approach assumes the existence of a function `REGRET()` that bounds the costs for scheduling a suboptimal operation instead of an optimal one. This bears the advantage that, for every sample, each operation can be rated according to its regret: eventually the operation minimizing the overall regret is scheduled on a broader information basis. The regret function, on the other hand, has to be computable in  $\mathcal{O}(f_{\text{Sched}})$ , where  $f_{\text{Sched}}$  refers to the time complexity of the underlying scheduler. Otherwise the algorithms' time complexity would increase beyond  $\mathcal{O}(f_{\text{Sched}}/|T(t)|)$ .

The pseudocode of the Regret algorithm is given in Figure 4, where `COMPUTESCHEDULEGA( $O$ )` invokes the optimizing GA scheduler for the operations in  $O$ , `EVAL( $s$ )` denotes the evaluation of the

<sup>1</sup>Slack time is defined as the time to deadline less the remaining processing and necessary delay times.

schedule  $s$  according to the chosen optimization criterion and  $\text{sort}(O, r)$  returns a list of elements  $o \in O$  in ascending order of the value  $r(o)$ .

The main difference between our algorithm and the original is a specialized seeding mechanism that includes the best 20% chromosomes from the previous optimization. We observed a reduction of the search costs for solutions of the same quality. This is consistent with results from [13]. The regret of an operation has been estimated very roughly by assuming its global left shift (including its unfinished predecessors). Hence, the regret  $r(o)$  of an operation  $o$  is given as

$$r(o) = p_o + \sum_{o' \in I} p_{o'},$$

where  $p_o$  denotes the (remaining) processing time of operation  $o$  and  $I$  denotes the set of all unfinished predecessor operations of  $o$  in job  $j_o$ .

## 5 Experimentation

In order to evaluate the performance of the schedulers introduced in Section 4, we have generated random problems with varying system utilization and variance in the stochastic variates. The generation process is described in Section 5.1, whereas the results are discussed in Section 5.2.

### 5.1 Setup

We found that most randomly selected problems are either too easy or instable. To generate “difficult” problems in the sense that they are close to being schedulable on time but to experience lateness in trivial first-come-first-served case, we proceeded in the following way.

Given the number of job classes  $n$  and the number of operations per job  $m$ ,  $m$  resource classes  $\{1, \dots, m\}$  with  $n$  resources are generated. Afterwards,  $n$  sets of  $m$  operation types  $O_j = \{o_{j,r} \mid 1 \leq r \leq m\}$ ,  $1 \leq j \leq n$ , are created, where  $o_{j,r}$  uses one resource of class  $r$ . Additional resource usage is randomly associated by drawing the number of additional resources from a  $\text{geom}(\frac{3}{m})$ -distribution and their classes from a  $\text{unif}\{1, \dots, m\}$ -distribution. The operations in  $O_j$  are then assigned to job  $j$  in a random permutation. This ensures a mean  $\frac{m}{3}$  operations with more than one resource—a moderate degree of competition among the job classes.

In our simulations, we assumed fixed transition times and precedence delays, exponentially distributed interarrival times as well as truncated normal distributions for the processing times and due dates. The number of job classes was fixed to 4 and the number of operations per job class to 6. The mean processing times were uniformly chosen from the interval  $[3.0, 10.0]$ , precedence and transition times were fixed to one third of its mean. Due dates were set to three times a job’s mean processing time (including delays). Both the variances and arrival rates were varied throughout the simulations.

We created four branches of scenarios with increasing maximum resource utilization  $\rho \in \{0.50, 0.65, 0.78, 0.95\}$  and increasing processing time variation coefficient  $c = \frac{\sigma}{\mu} \in [0, 1]$ .<sup>2</sup> The simulation length was chosen to allow approximately 10 000 job arrivals and runs were repeated independently 10 times. Hence, the size of the 95% confidence interval  $I$  for the lateness can be estimated by  $|I| \approx \frac{c + \Delta t}{100} \cdot \mu$ , where  $\Delta t$  is the mean time between arrivals. The number of samples for the Regret scheduler was adaptively set to  $(1 + 5 \cdot c)$  with 10 generations per sample in the genetic algorithm.

### 5.2 Results and Discussion

In the following, the results conducted from the experiments will be summarized. Figure 5 and Table 1 provide exemplary data.

---

<sup>2</sup> $\mu$  denotes the mean of a processing time and  $\sigma$  its standard deviation.

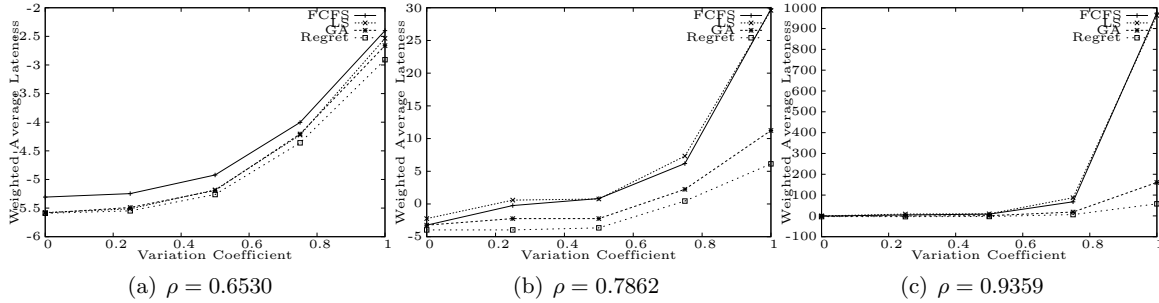


Figure 5: Weighted average lateness versus the estimated variation coefficient  $c = \frac{\sigma}{\mu}$  at resource utilization  $\rho$ .

Scheduler	Mean computation time per schedule					
	$c = 0$	$\rho = 0.6530$ $c = 0.5$	$c = 1$	$c = 0$	$\rho = 0.9359$ $c = 0.5$	$c = 1$
GA	$3.206 \cdot 10^{-3}$	$3.894 \cdot 10^{-3}$	$8.265 \cdot 10^{-3}$	$3.908 \cdot 10^{-2}$	$4.366 \cdot 10^{-2}$	$5.467 \cdot 10^{-1}$
Regret	$9.858 \cdot 10^{-3}$	$1.120 \cdot 10^{-2}$	$2.275 \cdot 10^{-2}$	$8.733 \cdot 10^{-2}$	$7.201 \cdot 10^{-2}$	$8.834 \cdot 10^{-1}$

Table 1: Mean computation time per schedule.

Averaged over all our simulations, the weighted average lateness performance criterion significantly outperformed maximum lateness in more than half of the runs (to a confidence of 95%, with lower differences in high utilization scenarios). Nevertheless did the schedulers using different optimization criteria exhibit similar behavior, hence the results concerning the maximum lateness criterion are omitted.

For the cases with a maximum utilization  $\rho < 0.6$ , both dispatching rules and optimization algorithms perform equally well for all variations of  $c$ . This is almost surely due to the lack of resource competition.

In the more interesting case of high utilization,  $0.6 \leq \rho < 1.0$ , optimization of the schedule bears a significant improvement of the computed schedules. We hypothesize that the improvement grows quadratically in the variation coefficient  $c$ , as the queue length in  $M/G/1$  models in steady state is  $2 \frac{\rho^2}{1-\rho} (1 + \sigma^2 \mu^2) = k(\rho) \cdot (1 + c^2 \mu^4)$ , where  $k(\rho)$  is a constant depending on  $\rho$ . Indeed, this hypothesis also provides an explanation for the rapid decay in the quality of the solutions computed by the dispatching schedulers with growing utilization. Thus, we conclude that despite (polynomially growing) runtimes<sup>3</sup> of the GA and Regret algorithms, their application is feasible because even under strong time constraints. For example, an optimization constrained to a runtime of 1ms seeded with results from dispatching algorithms showed mean improvements of 10% ( $c = 0.0$ ) to 30% ( $c = 1.0$ ) in another series of simulations.

Additionally the plain genetic algorithm is still outperformed by the Regret scheduling algorithm despite its coarse regret function (global left-shift) in the more relaxed setting with stochastic timings and precedence constraints. Actually this improvement grows with increasing variance of the processing times. This can be accounted to the structure and growing size of the solution space [13]. The GA scheduler cannot sufficiently search that space in the limited amount of time anymore and might get stuck in suboptimal solutions. The Regret scheduler on the other hand searches a different but still structurally similar search space for every sample. These repetitions allow to escape suboptimal solutions and improve the optimization results. However, this statement remains true only in the case of an adaptive number of samples. Otherwise misjudgments become

<sup>3</sup>The time complexity of our hybrid Giffler-Thompson algorithm is in  $\mathcal{O}(c_R \cdot T(t)^2 \log T(t))$ , where  $c_R$  is constant that depends on  $R$ . The runtime of the genetic algorithm remains constant beyond a certain problem size.

likely and the performance of the Regret scheduler degrades. To test this hypothesis, we fixed the number of samples while increasing  $c$  up to 2. The resulting performance degraded stepwise, even beyond the performance of the GA-scheduler.

## 6 Conclusion

We compared schedule quality obtained from dispatch and optimization schedulers under varying coefficient of variation and resource utilization. From our results, it can be concluded that the application of an optimizing scheduler is feasible as soon as any notable resource competition appears, particularly as the optimization algorithms can be adapted to the system's time requirements or system-specialized search heuristics. Even in our setting with loose due dates, this limit was reached with a fairly low utilization of  $\rho = 0.6$ .

Furthermore, we showed that in job shop problems with stochastic processing times, a regret-based scheduler approach retains the ability to improve over plain optimization algorithms, regardless of the utilization or variation (in the case of an adaptive number of samples). According to simulation results only briefly mentioned herein, we account this to the structure of the search space (cp. Watson et al. [13]). We suspect that this observation only holds for the case of moderate resource competition. An optimized resource allocation might increase a schedule's accuracy beyond the gain of the Regret approach if the difficulty of resource allocation increased further.

Currently a study on this supposition is subject to our research. In further work we want to investigate whether it is possible to combine advantages of the scheduling techniques using a Markov Decision Process as it has been done in [4] with different dispatching rules.

## References

- [1] R. Bent, P. Van Hentenryck (2004), Regrets Only! Online Stochastic Optimization under Time Constraints, *Proc. of the 19th National Conference on Artificial Intelligence*, 501 – 506.
- [2] R. Bent, P. Van Hentenryck, E. Upfal (to appear), Online Stochastic Optimization Under Time Constraints,
- [3] C. Bierwirth, D.C. Mattfeld (1999), Production Scheduling and Rescheduling with Genetic Algorithms, *Evolutionary Computing* **7**(1), 1 –17 .
- [4] H.S. Chang, R. Givan, E.K.P. Chong (2000), On-line Scheduling via Sampling, in *Artificial Intelligence Planning Systems*, 62 – 71.
- [5] M.P.J. Fromherz (2001), Constraint-based Scheduling, in *American Control Conference*, Arlington, VA.
- [6] W.S. Gere (1966), Heuristics in Job Shop Scheduling, *Management Science* **13**, 167 – 190.
- [7] S.K. Goyal, K. Mehta, R. Kodali, S.G. Deshmukh (1996), Simulation for Analysis of Scheduling Rules for a Flexible Manufacturing System, *Integrated Manufacturing Systems* **6**(5), 21 – 26.
- [8] D. Montana (2005), A Comparison of Combinatorial Optimization and Dispatch Rules for Online Scheduling, *Proc. of the 2nd Multidisciplinary Conference on Scheduling: Theory and Application*, 353 – 362.
- [9] S. Panwalker, W. Iskander (1977), A Survey of Scheduling Rules, *Operations Research* **25**(1), 45 – 61.

- [10] thesis N. Policella (2005), Scheduling with Uncertainty - A Proactive Approach Using Partial Order Schedules, University of Rome "La Sapienza", Ph. D. Thesis.
- [11] S.F. Smith (1994), OPIS: A Methodology and Architecture for Reactive Scheduling, 29 – 66, M. Zweben and M. Fox eds, *Intelligent Scheduling*, Morgan Kaufmann.
- [12] R.-L. Sun, H.-X. Li, Y. Xiong (2006), Performance-Oriented Integrated Control of Production Scheduling, *IEEE Transactions on Systems, Man, and Cybernetics* **36**(4), 554 – 562.
- [13] J.-P. Watson, J.C. Beck, A.A. Howe (2003), Problem Difficulty for Tabu Search in Job-Shop Scheduling, *Artificial Intelligence*.