Slack-based Techniques for Robust Schedules

Andrew J. Davenport[1], Christophe Gefflot[2], and J. Christopher Beck[2]

[1] IBM T J Watson Research Center, PO Box 218, Yorktown Heights
New York 10598 USA
davenport@us.ibm.com

[2] ILOG, S.A., 9, rue de Verdun, B.P. 85, F-94523 Gentilly Cedex France
{cgefflot, cbeck}@ilog.fr

Abstract. Many scheduling systems assume a static environment within which a schedule will be executed. The real world is not so stable: machines break down, operations take longer to execute than expected, and orders may be added or canceled. One approach to dealing with such disruptions is to generate robust schedules: schedules that are able to absorb some level of unexpected events without rescheduling. In this paper we investigate three techniques for generating robust schedules based on the insertion of temporal slack. Simulation-based results indicate that the two novel techniques out-perform the existing temporal protection technique both in terms of producing schedules with low simulated tardiness and in producing schedules that better predict the level of simulated tardiness.
Keywords: Robustness, Uncertainty, Scheduling, Heuristics

1 Introduction

Based on a field study of a number of job shops, McKay et al. [MSB88] comment that "the [static job shop] problem definition is so far removed from job-shop reality that perhaps a different name for the research should be considered". In particular, they found that modern scheduling technology failed to adequately address scheduling in uncertain, dynamic environments.

There are two general approaches to dealing with uncertainty in scheduling. Whereas *reactive* techniques address the problem of how to recover from a disruption once it has occurred, *pro-active* scheduling constructs schedules that account for statistical knowledge of uncertainty. One way of achieving this is by generating *robust* schedules, that is, a schedule with "the ability to satisfy performance requirements predictably in an uncertain environment" [LP91].

In this paper, we explore slack-based techniques for robust scheduling. The central idea behind slack-based techniques is to provide each activity with extra time to execute so that some level of uncertainty can be absorbed without rescheduling. We define the amount slack for an activity, A, as follows:

$$slack_A = lft_A - est_A + dur_A \tag{1}$$

Where est_A and lft_A are respectively the earliest start time and latest finish time of activity A and dur_A is the duration of activity A.

Three slack-based techniques are examined in this paper. The first, *temporal protection* [Gao95], adds slack to an activity before scheduling. The original duration of each activity is extended and then this protected duration is used during scheduling. Two novel techniques are introduced here:

- Time window slack (TWS): Rather than extending the durations of activities, this technique modifies the problem definition to ensure that each activity will have at least a specified amount of slack. The advantage of this approach over temporal protection is that the amount of slack for each activity can be reasoned about during the problem solving rather than being "hidden" inside the protected duration.
- Focused time window slack (FTWS): TWS and temporal protection specify the amount of slack required for each activity before problem solving. In FTWS, the amount of slack for each activity depends on where along the temporal horizon an activity is scheduled. The intuition is that the later in the schedule an activity is executed, the more likely it is to have a disruptive event occur before its execution and, therefore, the more slack is needed.

2 Problem Definition

The problem addressed here is the job shop scheduling problem with release and due dates, machine breakdowns, and the optimization criteria of minimization of the sum of job tardiness.

Each job is composed of a set of totally ordered activities. Each job, j, has a release date, $rd(j)$, and a due date, $dd(j)$. The former is the earliest time an activity in the job can execute and the latter is the latest time that the last activity in the job should finish execution. Each activity requires a single resource (also referred to as a machine) and has a predefined duration during which it must be the only activity executing on its required resource. Once an activity has begun execution it cannot be pre-empted by another activity.

The goal is to sequence the activities on each resource such that the order within each job is respected and that the sum of the job tardiness is minimized. More formally, given $C(j)$, the completion time for the last activity in job j, we seek to minimize $\sum \max(0, C(j) - dd(j))$ over all jobs in a problem.

To represent uncertainty, some machines are subject to breakdowns. During a breakdown, a machine cannot process any activities and any activity which was executing on the machine at the time of breakdown is stopped and then resumed from the point where it was stopped after the machine has been repaired. Statistical characteristics of machine breakdowns are known.[1] We assume normal distributions parameterized by:

- $\mu_{tbf}(R)$: the mean time between failure of resource R
- $\sigma_{tbf}(R)$: the standard deviation in time between failure on R
- $\mu_{dt}(R)$: the mean down time (or duration of breakdown) on R
- $\sigma_{dt}(R)$: the standard deviation in down time on R

[1] In a production scheduling domain, such characteristics may be supplied by the manufacturer and/or may be based on shop floor operational history.

3 Temporal Protection

Temporal protection [Gao95] is a preprocessing technique which extends the duration of each activity based on the uncertainty statistics of the resource on which it executes.[2]

Resources that have a non-zero probability of breakdown are designated as *breakable* resources. The durations of activities requiring breakable resources is extended to provide extra time with which to cope with a breakdown. The scheduling problem with protected durations is then solved with standard scheduling techniques.

The intuition behind the extension of durations is that during schedule execution, the protected durations provide slack time which can be used in the event of machine breakdown. For instance, in Figure 1, activities A and B are sequenced to execute consecutively on a breakable resource. The length of the white box represents the original duration of the activities while the shaded box represents the extension due to temporal protection. If the machine breaks down while activity A is executing, the extra time within the protected duration can be used to absorb the breakdown. If the breakdown lasts no longer than the available protection, its effect will not be felt in the rest of the schedule. If the breakdown lasts longer, some reactive approach must be taken to restore consistency to the schedule. If no breakdown occurs during the execution of activity A, then activity B can start earlier: the slack provided by the temporal protection for A is available for use by activity B.

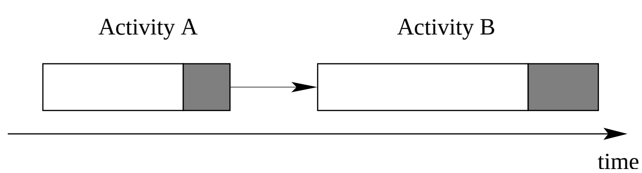

Fig. 1. Example of a temporally protected schedule, with white boxes representing the original duration and grey boxes representing the extended durations.

A key issue is the amount of temporal protection added to each activity. Too much protection will result in a poor quality but highly robust schedule. Too little protection will also result in a poor quality schedule execution if a breakdown occurs. The approach taken by Gao extends the duration so as to amortize the breakdown over a number of activities.

More formally, given an activity, A, that requires a breakable resource, R, temporal protection defines a protected duration for A, $dur_{A,tp}$, where tp denotes temporal protection, as follows:

[2] We present a simplified description of temporal protection in the case that each activity requires only one resource. For a formulation for multiple resource requirements see [Gao95].

$$dur_{A,tp} = dur_A + \frac{dur_A}{\mu_{tbf}(R)} \times \mu_{dt}(R) \quad (2)$$

The equation specifies that the protected duration of an activity A is its original duration plus the duration of breakdowns that are expected to occur during the execution of A.

4 Time Window Slack

By extending activity durations in a preprocessing step, temporal protection transforms the original problem to a new problem that can be solved with any scheduling technique. We conjecture, however, that the preprocessing has a disadvantage in that during scheduling the amount of slack added to each activity cannot be directly reasoned about. This can lead to situations where it is impossible to share slack between activities even when no resource breakdowns have occurred. Indeed, the claim that activity B in Figure 1 can start execution early if there is no breakdown during or before activity A is an over-simplification. For example, in Figure 2 we introduce a third activity, C, which executes on a non-breakable resource and so has no temporal protection. Activity B must execute after activity C, and since activity C finishes execution later than activity A, the earliest start time of B is the end time of C. The temporal protection represented by the extension of the duration of activity A is not available for use by activity B as B cannot start earlier than the end time of C.

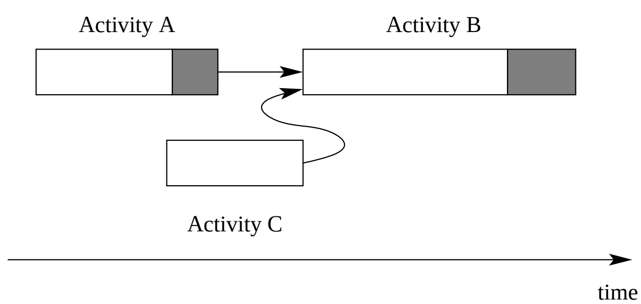

Fig. 2. A situation where the temporal protection cannot be shared between activities A and B.

To avoid such situations, we propose the *time window slack* (TWS) approach which reasons directly about the slack time available for an activity during problem solving. Rather than including this slack as part of the activity duration, we explicitly reason about it by adding a relation to the problem definition that specifies that schedules must have sufficient slack time for each activity.

The advantage of this approach is that there is more information about activity slack during the problem solving. In a situation such as the one in Figure 2, we are able to detect that the slack of activity A cannot be shared with activity B. If there is still sufficient slack in the schedule after activity B, we still may be able to generate a valid schedule. If not, we must backtrack and continue search.

The amount of slack for each activity is still critical for the generation of a robust schedule. Using $acts_R$ to denote the set of all activities executing on resource R, the required slack for activity $A \in acts_R$ is:

$$slack_A \geq \frac{\sum_{B \in acts_R} dur_B}{\mu_{tbf}(R)} \times \mu_{dt}(R) \tag{3}$$

The required slack for an activity under TWS is considerably larger than the duration extension in temporal protection. Indeed, the amount of slack on each activity is equal to the sum of the durations of all the expected breakdowns on R. This difference is because we expect the slack on all activities on a resource to be shared. If the slack is completely shared then the total slack on a breakable resource is approximately equal (given integer rounding) to the sum of the durations extensions in temporal protection.

The relation in inequality 3 is somewhat problematic for standard scheduling approaches: no solution which assigns start times to activities can satisfy it unless the left-hand side evaluates to 0. The very act of assigning a start time forces the slack (as defined in equation 1) to be 0. Therefore, rather than being able to use arbitrary scheduling techniques, we must use scheduling techniques that reason about the order of activities on resources. Fortunately, such techniques are not uncommon (e.g., [SC93,BF00]).

5 Focused Time Window Slack

Neither temporal protection nor TWS take into account the placement of activities on the scheduling horizon. For example, consider scheduling a newly repaired machine whose $\mu_{tbf}(R)$ is 1000 days and whose $\sigma_{tbf}(R)$ is 50 days. Given the negligible probability of a breakdown before day 800, it does not seem worthwhile to focus on making the schedule more robust before this time. *Focused time window slack* (FTWS) uses the uncertainty statistics to focus the slack on areas of the horizon that are more likely to need it to deal with a breakdown.

The probability distribution, $P(N(\mu_{tbf}(R), \sigma_{tbf}(R)) \leq t)$, allows us to compute the probability that a breakdown event will occur at or before time t. An approximation of this curve can be efficiently computed using standard statistical techniques. This curve is used to determine the amount of slack an activity should have given the basic intuition that the higher the probability of a breakdown occurring before the execution an activity, the greater the amount of slack.

The slack for an activity is computed as a function of the probability that a breakdown will occur before or during the execution of the activity and of the expected breakdown duration. If the $\mu_{tbf}(R)$ for a machine is much less than the scheduling horizon, the possibility of multiple breakdowns must be considered. We do this by considering the cases for each breakdown, nb, separately. First, we assume that at time 0 the machine has just been maintained. For each value of $nb = 1..M$, where M is a large number, we compute the expected time that the nb-th breakdown will occur as:

$$\mu(nb) = (nb \times \mu_{tbf}(R)) + ((nb-1) \times \mu_{dt}(R)) \tag{4}$$

We calculate the standard deviation of the time for nb breakdowns as:

$$\sigma(nb) = ((nb \times \sigma^2_{tbf}(R)) + ((nb-1) \times \sigma^2_{dt}(R)))^{\frac{1}{2}} \tag{5}$$

These calculations constitute an abuse of the central limit theorem: the random variables representing the breakdown events are not independent. This calculation is an approximation and future work will examine a more sophisticated statistical analysis.

We use the statistics for nb breakdowns to calculate the $P(N(\mu(nb), \sigma(nb)) \leq t)$ curve estimating the probability that nb breakdowns will occur before a particular time t. The amount of slack time required for an activity executing at a particular time point t on resource R is:

$$slack_A(t, R) \geq \sum_{nb=1}^{M} P(N(\mu(nb), \sigma(nb)) \leq t) \times \mu_{dt}(R) \tag{6}$$

As with TWS, we add relation 6 to the problem model as a pruning rule.

6 Experimental Evaluation

To evaluate the three robustness techniques we run a simulation-based experiment. Each problem is solved to optimality: an ordering of the activities on each resource is found that minimizes the sum of the tardiness of the jobs. A simulation of the "execution" of each schedule under uncertainty is then performed.

The problem sets used in our experiments were constructed as follows. Ten 6×6 job shop problems with uncorrelated durations were constructed using a job shop problem generator [WBHW99]. For each problem, TLB, the lower bound on the makespan due to Taillard [Tai93], was calculated. The release dates for each job were assigned by randomly choosing a time (with uniform probability) from the interval $[0, \frac{TLB}{8}]$. Standard temporal propagation was then performed to provide a lower-bound, $dd_{lb}(j)$, on the due date of each job. For each original problem, six problems were then generated by setting the actual due date of each job to $dd(j) = dd_{lb}(j) \times L$, where L represents the "looseness" of the due dates and ranges from 1.0 to 1.5 in steps of 0.1.

For each of these 60 problems, we then introduced nine levels of uncertainty based on two *uncertainty factors*: $U_{machine}$, the number of machines prone to failure and U_{stat}, the magnitude of the uncertainty statistics. Each of the uncertainty factors have three levels $\{1, 2, 3\}$ and the overall level of uncertainty is, $U = 3 \times (U_{machine} - 1) + U_{stat}$. This encoding produces nine levels of uncertainty divided into three groups. Levels 1-3 have one breakable machine, levels 4-6 have two breakable machines, and levels 7-9 have 3 breakable machines. Within each group the likelihood of breakdown on each breakable machine increases as the level increases.

The statistical values for each level of U_{stat} are derived as follows for a breakable resource, R. Given the set of activities, $acts_R$, requiring resource R, a lower-bound, $lb(R)$ on the latest end time of the activities is calculated:

$$lb(R) = \max(\min_{A \in acts_R}(est_A) + \sum_{A \in acts_R} dur_A, \max_{A \in acts_R}(lft_A)) \tag{7}$$

Using $lb(R)$, we define, $\mu_{tbf}(R, U_{stat})$, the mean time between failure for resource R, and $\sigma_{tbf}(R)$, the standard deviation time between failure, as follows:

$$\mu_{tbf}(R, U_{stat}) = \frac{lb(R)}{2^{U_{stat}-1}}, \ \sigma_{tbf}(R) = \frac{lb(R)}{8} \tag{8}$$

The standard deviation of the down time $\sigma_{dt}(R)$ is simply the mean duration of the activities in $acts_R$ while the mean down time, $\mu_{dt}(R)$ is twice that value.

As we began with 10 problems, 6 values for L, the due date looseness factor, and have a total of 9 combinations of the uncertainty factors, we have a total of 540 test problems.

6.1 Evaluation Criteria

The evaluation of the schedules under uncertainty is done using a simulator. Our optimization function, therefore, has two forms: simulated and predicted. Given problem instance, p, we use $TARD(p, *)$ to denote the minimal sum of the tardiness over all jobs in a predictive schedule. Similarly, we use $TARD(p, s)$ to denote the tardiness of problem instance p in simulation s.

Given a set of simulations, S, and a set of problems, P, the primary basis of comparison of our robustness techniques is the mean simulated tardiness:

$$MST(P, S) = \frac{\sum_{s \in S, q \in Q} TARD(p, s)}{|S| \times |Q|} \tag{9}$$

Our secondary evaluation criteria is the mean absolute difference between the predicted tardiness and the simulated tardiness.

$$MATD(P, S) = \frac{\sum_{s \in S, q \in Q} |TARD(p, s) - TARD(p, *)|}{|S| \times |Q|} \tag{10}$$

6.2 Results

For each test problem and for each robustness technique (including the *no protection* where the uncertainty statistics were ignored), each scheduling problem was solved to optimality using ILOG OPL Studio 3.1, ILOG Scheduler 4.4, and ILOG Solver 4.4. Solving a single problem to optimality took approximately 10 seconds, regardless of robustness technique, on a Pentium II, 300 MHz PC.

A simulator, written in ILOG OPL Studio 3.1, simulated the execution of each schedule, introducing breakdowns based on the specified uncertainty distributions. When a breakdown occurred, the duration of the executing activity was extended by the duration of the breakdown. In the temporal protection condition, the protected durations were replaced with the original durations and the activities were left-shifted (subject to their release dates) before the simulation.

Figure 3 presents the graph of, $MST(P, S)$, the mean simulated tardiness for 100 simulations of each problem under each robustness technique and each combination of uncertainty factors. Except for the highest level of uncertainty, temporal protection results in a higher mean tardiness than is observed even if the uncertainty information is ignored. This is consistent with previous experiments with temporal protection [Gao95]. In contrast, both TWS and FTWS achieve a lower mean tardiness than no protection across all uncertainty levels with FTWS achieving slightly lower mean tardiness than TWS.

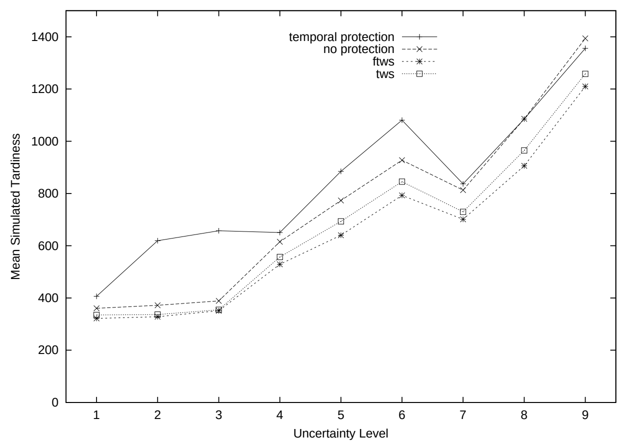

Fig. 3. The mean simulated tardiness for each uncertainty level

Figure 4 presents, $MATD(P, S)$ the mean difference in the absolute value between the simulated tardiness and the predicted tardiness for each robustness technique. Here we observe that for low levels of uncertainty, the predictions of the TWS, FTWS, and no protection techniques are quite similar. In contrast, the temporal protection results vary widely: the mean absolute difference is four times greater than that of the other techniques at uncertainty level 3. As the level of uncertainty increases however, we see the mean absolute difference for no protection increasing quickly while TWS and FTWS results increase more slowly. Interestingly, the relative results of temporal protection improve significantly with increased uncertainty, achieving the lowest mean absolute difference of all techniques at uncertainty levels 7 through 9.

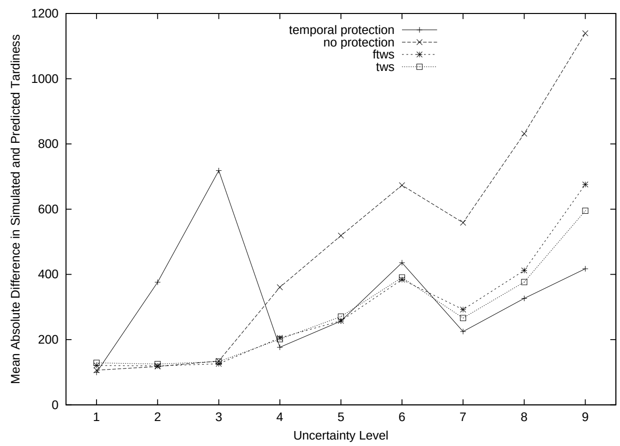

Fig. 4. The mean absolute difference between simulated and predicted tardiness for each uncertainty level

7 Discussion

There are two goals for robustness techniques in scheduling. The first is that though in building robust schedules, the overall schedule quality may be diminished, the accuracy of the predictive schedules is increased. The ability to better predict the actual completion time of a job, even if this completion time is tardy is valuable in real world scheduling. The second goal is that by taking uncertainty into account, the predictive schedule will not only provide more accurate performance information but will actually result in better overall schedule performance. This better performance comes from the fact that the predictive schedule can actually be constructed using the uncertainty information.

In comparing the robustness techniques with ignoring uncertainty information, we see that all techniques achieve the first goal with the exception of temporal protection at low levels of uncertainty. At uncertainly levels above level 3, the absolute difference between the simulated and predicated tardiness (Figure 4) is approximately two times smaller when the uncertainty is taken into account. These differences are not apparent at lower levels of uncertainty and, indeed, temporal protection performs very badly at level 3.

TWS and FTWS also achieve the second goal: Figure 3 shows that the simulated tardiness for the TWS and FTWS solutions is less than that for the solutions with no protection. Except for high level of uncertainty, temporal protection does not result in better overall schedules.

Looking more deeply at the experimental results, we see two interesting phenomenon. First, when only one machine is breakable (i.e., levels 1-3) the no protection condition performs almost as well (and in some cases better) than TWS and FTWS on both mean simulated tardiness and mean absolute difference measures. This is not terribly surprising as, at low levels of tardiness, breakdowns are less disruptive: unless the breakdown occurs during an activity that is on a critical path[3] some of the breakdown will be absorbed by the naturally occurring slack. Furthermore, with low levels of uncertainty, the level of slack required in the TWS and FTWS conditions is small. The activity sequences in the optimal solutions in the no protection condition will therefore be quite similar to those of TWS and FTWS. Similar sequences will lead to similar simulated tardiness results.

The second phenomenon is that while temporal protection performs very poorly in terms of absolute difference with one breakable machine, when three machines are breakable it has a lower mean absolute difference than TWS and FTWS. The poor behavior, especially at level 3, arises from the fact that extending the durations of the activities on the breakable resource leads to a scheduling problem where the breakable resource is essentially a bottleneck. The optimality of a solution depends almost wholly on the sequence of activities on that resource while the sequences on the rest of the resources are irrelevant. An optimal solution, therefore, has an almost random sequencing of activities on the non-breakable resources. When the duration extensions are removed in the simulation, the sub-optimal sequences on the non-breakable resources leads to high tardiness. In contrast, with multiple breakable machines, optimality depends on more than one resource, leading to a better sequence of activities over more of the resources. The TWS and FTWS methods do not lead to a single bottleneck resource when there is only one breakable machine. This is because the slack that is added to the activities on the breakable resource affects upstream and downstream activities as well. Since by construction all jobs have one activity on the breakable resource, all activities in the problem are constrained to have some level of slack. Even though there is only one breakable resource, all resources are required to have an equal amount of slack and therefore there is no bottleneck resource that completely defines optimality. During problem solving, the activity sequences on the non-breakable resources are just as important as those on the breakable resource in terms of optimality. The fact that the slack is "propagated" to activities that are not on a breakable resource is, in retrospect, obvious. Based on our results, however, it may have a significant impact on the performance of robustness techniques. An interesting question arises as to the relative contribution of reasoning about slack during problem solving and of "slack propagation" toward dealing with uncertainty.

We do not as yet have an explanation of the good performance of temporal protection with high levels of uncertainty.

7.1 Relation to Previous Work

Slack-based techniques involve the addition of extra time in order to recover from unexpected events. Similar approaches, called temporal redundancy, are common in real-time fault tolerant scheduling [GMM95]. Such scheduling problems differ from those typically investigated in the AI community both in the scope (i.e.,

[3] See [Kre00] for a definition of critical path on tardiness minimization problems.

often only one machine) and in the definition of a solution (e.g., a guarantee that the system is schedulable). Nonetheless, real-time fault tolerant scheduling research represents an important source of ideas for further investigations.

Overall, there has been little work in the research literature that specifically addresses uncertainty in the context of the types of scheduling problems that are typically of interest in AI (e.g., problems with multiple resources and activities). A variety of techniques, including resource redundancy [GMM95], probabilistic reasoning [BPLW97,DC97], and a variety of off-line/on-line approaches [Ber93,GB97,Hil94,MHK+98] have been investigated, usually in the context of simpler scheduling problems. There does not yet appear to be a broader understanding of either the role that uncertainty plays in real scheduling problems or a comparison of different approaches.

8 Conclusion

In this paper, we examined three techniques for taking into account uncertainty in scheduling by adding slack to the scheduling problem. Our experiments demonstrate that an existing technique, temporal protection, results in a reduced overall schedule performance but more accurate schedules than not taking uncertainty information into account. The sole exception is at low levels of uncertainty, when temporal protection produces schedules that are significantly less accurate than no protection.

Two novel techniques, time window slack and focused time window slack, were developed to account for the fact that temporal protection reasons about uncertainty as a preprocessing step, before actual scheduling. Time window slack and focused time window slack both incorporate reasoning about uncertainty into the problem solving as well as resulting in a propagation of slack time from activities on breakable resources to temporally connected activities. Our experiments indicate that both the novel techniques are able to produce better, more accurate schedules than either temporal protection or no protection.

We view the work reported in this paper as preliminary. As noted above, there are a number of approaches to uncertainty that have been tried in various types of scheduling problems, however there is not, as yet, any broader understanding of uncertainty as it applies to scheduling problems typically investigated in the AI literature. This paper demonstrates that for a simple, but interesting, class of scheduling problems, slack-based techniques can provide higher quality, more accurate schedules.

9 Acknowledgments

Portions of this research were funded by the Materials and Manufacturing Council of Ontario. Part of this research was performed while the first author was a visiting researcher at SINTEF Applied Mathematics. We would like to thank Geir Hasle, Dag Kjenstad and Martin Stolevik for useful discussions.

References

[Ber93] P. M. Berry. Uncertainty in scheduling: Probability, problem reduction, abstractions, and the user. In *IEE Colloquium on Adcanced Software Technologies for Scheduling*, 1993. Digest No: 193/163.

[BF00] J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.

[BPLW97] A Burns, S. Punnekkat, B. Littlewood, and D.W. Wright. Probabilistic guarantees for fault-tolerant real-time systems. Technical Report DeVa TR No. 44, Design for Validation, Esprit Long Term Research Project No. 20072, 1997. Available at http://www.fcul.research.ec.org/deva.

[DC97] R.L. Daniels and J.E. Carrillo. β-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29:977–985, 1997.

[Gao95] H. Gao. Building robust schedules using temporal protection–an empirical study of constraint based scheduling under machine failure uncertainty. Master's thesis, Department of Industrial Engineering, University of Toronto, 1995.

[GB97] R.P. Goldman and M.S. Boddy. A constraint-based scheduler for batch manufacturing. *IEEE Expert*, 12(1):49–56, 1997.

[GMM95] S. Ghosh, R. Melhem, and D. Mossé. Enhancing real-time schedules to tolerate transient faults. In *Real-Time Systems Symposium*, 1995.

[Hil94] D. W. Hildum. *Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems.* PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, MA. 01003-4610, 1994. UMass CMPSCI TR 94-77.

[Kre00] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3), 2000.

[LP91] C. Le Pape. Constraint propagation in planning and scheduling. Technical report, CIFE Technical Report, Robotics Laboratory, Department of Computer Science, Stanford University, 1991.

[MHK+98] N. Meuleau, M. Hauskrecht, K.E. Kim, L. Peshkin, L.P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[MSB88] K.N. McKay, F.R. Safayeni, and J.A. Buzacott. Job-shop scheduling theory: What is relevant? *Interfaces*, 18(4):84–90, 1988.

[SC93] S. F. Smith and C. C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 139–144, 1993.

[Tai93] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[WBHW99] J.P. Watson, L. Barbulescu, A.E. Howe, and L.D. Whitley. Algorithms performance and problem structure for flow-shop scheduling. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 688–695, 1999.