# A Survey of Techniques for Scheduling with Uncertainty

Andrew J. Davenport

*IBM T J Watson Research Center*
*PO Box 218*
*Yorktown Heights*
*New York 10598 USA*
E-mail: davenport@us.ibm.com

J. Christopher Beck

*ILOG, SA*
*9 rue de Verdun*
*94253 Gentilly Cedex, France*
E-mail: cbeck@ilog.fr

Many scheduling systems assume complete information about the problem to be solved and a static environment within which the schedule will be executed. The real world is not static: machines break down, activities take longer to execute than expected, and orders may be added or canceled. In this article we present a survey research performed within the Artificial Intelligence and Operations Research communities concerning the management of uncertainty in scheduling.
**Keywords:** uncertainty, scheduling, survey, artificial intelligence, operations research

## 1. Introduction

Scheduling concerns the allocation of resources to activities over time in order to achieve some goals: produce some raw material that has been ordered, allow a class to be taught, or transport people or cargo from one location to another. Automated scheduling technology developed in the fields of Artificial Intelligence (AI) and Operations Research (OR) is increasingly being deployed to provide that scheduling functionality. However, scheduling in industry and particularly in the research literature is usually seen as a function of known,

perfect inputs. The set of orders, capacities of machines, duration of activities, and other characteristics of the scheduling problem are assumed to be known and static.

In a real world environment such as a factory, however, the probability of a precomputed schedule will be executed exactly as planned is low: machines malfunction, raw material deliveries are delayed, and resources are not available when required. Such disrupted execution incurs higher costs due to missed customer delivery dates, higher work-in-process inventory, and higher idle time for both people and machines. In a study of the job shop problem, McKay et al. claim that the dynamic characteristics of some real-world scheduling environments render the bulk of existing solution approaches unusable when applied to practical problems [32].

Uncertainty in scheduling may arise from many sources:

- machine breakdown;
- staffing/operator problems;
- unexpected arrival of new orders;
- cancellation or modification of existing orders;
- early or late arrival of raw materials;
- modification of release and due dates;
- uncertainty in the duration of processing of activities.

Since the real world is not static, decisions based upon rigid assumptions about real world behavior are not reliable. Hildum [24] notes that a schedule which is determined to be optimal prior to its execution is optimal only to the degree that the real world behaves as expected during the schedule's execution. As a result, in uncertain environments, it may not be practical to devote significant effort to achieving optimality, since the true optimality can only be ascertained in conjunction with its execution in the real world. An apparently optimal schedule may be based on an unreasonable set of expectations about the real world, and therefore may be significantly less than optimal when executed. Similarly, a schedule which appears less than optimal before execution but which contains some built-in flexibility for dealing with unexpected events, may turn out to be a good schedule upon execution.

The aim of this document is to survey the field of scheduling in the presence of uncertainty. We give a high level overview of the work carried out in scheduling

which deals in some way with uncertainty and describe a number of such systems in detail.

### *1.1. Common Models of Scheduling*

Informally, scheduling problems are composed of *activities*, *resources* and *constraints*[1]. An activity represents a process which uses resources to produce goods or provide services. Activities have a start time, end time and duration, specifying the period of time over which execution of the activity takes place. The two most important classes of constraints are *temporal constraints* and *resource constraints*. Temporal constraints express temporal relationships between activities, such as one activity must take place after another. Resource constraints express constraints on resource usage, such as a resource may process only one activity at a time. A full ontology of scheduling is outside the scope of this paper and can be found in Smith & Becker [40].

Two common models of scheduling which have been widely studied are the *Job Shop Scheduling Problem* and the *Resource Constrained Project Scheduling Problem*. We briefly review these two models in the remainder of this section. We use these perfect information models to provide a basis for understanding the ways in which reasoning about uncertainty has been introduced into scheduling.

An $n \times m$ job shop scheduling problem (JSSP) consists of $n$ jobs and $m$ resources. Each job consists of a set of $m$ completely ordered activities, where each activity has a duration for which it must execute and a resource which it must execute on. The complete ordering defines a set of precedence constraints, meaning that no activity can begin execution until the activity that immediately precedes it in the complete ordering has finished execution. Each of the $m$ activities in a single job requires exclusive use of one of the $m$ resources defined in the problem. No activities that require the same resource can overlap in their execution and once an activity is started it must be executed for its entire duration (i.e., no pre-emption is allowed). The job shop scheduling decision problem is to decide if all activities can be scheduled, given for each job a release date of 0 and a due date of the desired makespan, $D$, while respecting the resource and precedence constraints. The job shop scheduling decision problem is NP-complete [16].

A resource constrained project scheduling problem (RCPSP) consists of a

[1] In the Operations Research literature, activities are often referred to as *operations* or *tasks* and resources are often referred to as *machines*.

set of $n$ activities and $m$ resources. Each activity has a defined duration and may be linked by precedence constraints to any of the other activities. Each activity requires some amount of one or more of the $m$ resources during its duration of execution. Each resource has a maximum capacity expressing the total amount of the resource that can be used at any time point by any set of activities. As with the job shop problem, a solution to the decision problem is to determine, given a release date for all activities of 0 and a due date of the desired makespan, if there exists a schedule that respects the resource capacity constraints and the precedence constraints. The decision variant of the RCPSP is NP-complete [23].

More realistic models of scheduling problems include components of JSSP and RCPSP [36] as well as a variety of additional constraints (e.g., breaks during which a particular resource cannot be used, time dependent resource availability, set-up and tear-down activities required before each activity, etc.).

It should be noted that it is seldom the case that a schedule is executed in isolation. Real world schedules depend on and are depended on by external entities such as customers and suppliers in a supply chain of a manufacturing organization. A schedule therefore is not simply an internal recipe for a set of activities but also a basis for communication and coordination with external entities. These external dependencies make the management of uncertainty even more critical as unexpected events that are not reacted to and contained may have an impact that far out-weighs their original importance.

### *1.2. Uncertainty in Scheduling*

Two examples from the literature are presented in this section in order to provide concrete examples of uncertainty that arises in the execution of schedules.

#### *1.2.1. The Pathological Machine Shop*

As part of a larger analysis, McKay et al.[32] studied a factory for the machining of alloy castings. The scheduling problem contained approximately 80 activities per job, 300 work centers, and 5000 active jobs at any one time. When the study took place, all orders were behind schedule.

There are very many sources of uncertainty in this problem:

- Set-up time: the set-up time for the same part on the same machine with the same operator can vary from two days to six weeks.

- Release dates: the arrival of castings cannot be forecast accurately because of the alloys used and the difficulties in preparing the raw materials.
- Climatic sensitivity: machines are very sensitive to temperature and humidity, which affects their yield as well as their failure rate. A two degree ambient temperature shift can cause close tolerance parts to go in and out of specification while waiting for assembly.
- High priority orders: the practice of preempting jobs and pushing politically sensitive jobs through the shop throws the production cycle off and causes many jobs to be late. Senior management accepts orders and guarantees results without consulting shop floor schedulers.
- Overtime: the unionized work-force was accustomed to weekend overtime, and decreases productivity on Thursday and Friday to ensure weekend activity.

The conclusion of their study was that the theoretical formulation of job shop scheduling problem may be irrelevant: "it does not capture the essence of the job-shop scheduling problem faced by schedulers, and consequently, the research results have little applied value".

*1.2.2. Airport Ground Service Scheduling*

Another example of an uncertain scheduling environment is the *Airport Ground Service Scheduling Problem (AGSSP)* [24]. The AGSSP is defined as follows:

> **Definition 1:** We are given a master timetable of flights $\mathcal{F} : \{\mathcal{F}_\infty, \ldots, \mathcal{F}_\mathcal{I}\}$ and a collection of resources $\mathcal{R} : \{\mathcal{R}_\infty, \ldots, \mathcal{R}_\mathcal{R}\}$. Each of the flights in $\mathcal{F}$ requires the execution of some sequence of ground-servicing tasks from the activity set $\mathcal{T} : \{\mathcal{T}_\infty, \ldots, \mathcal{T}_\mathcal{T}\}$, depending on the particular type of ground service requested. A *job*, comprised of some sequence of $T_i$'s $(T_1, \ldots, T_n)$, is instantiated for each flight $F_i$ in $\mathcal{F}$. All flights have a ready time, corresponding to flight arrival time, and a due date, corresponding to flight departure time. The goal of the problem is to allocate resources to activities while satisfying the temporal and resource constraints.

Once again, there are many sources of uncertainty in this problem. The set of flights $\mathcal{F}$ is not fixed. Airport timetables are subject to fluctuation as flights are canceled, delayed or modified in the course of execution. Processing time is inherently dynamic. The duration of an activity may depend on the time of day

it is executed or on which resource is used to process the activity. Weather conditions, both local and remote, also have a significant impact on flights arriving and leaving the airport. Delays in timetables as a result of problems at other airports can also add to uncertainty. Many ground servicing tasks take longer to complete under difficult weather conditions, and may require extra, unplanned for activities to be executed in such situations.

### *1.3. Dealing with uncertainty*

In general, there are two approaches to dealing with uncertainty in a scheduling environment: *proactive* and *reactive* scheduling. Proactive scheduling constructs predictive schedules that account for statistical knowledge of uncertainty. Reactive scheduling involves revising or reoptimizing a schedule when an unexpected event occurs.

#### *1.3.1. Proactive scheduling*

The goal of proactive scheduling is to take into account the uncertainty in forming the original predictive schedule. The consideration of uncertainty information is used to make the predictive schedule more *robust*. A robust schedule has been defined as:

- one that is likely to remain valid under a wide variety of disturbances [31];
- one where "the violation of the assumptions upon which it is built are of no or little consequence" [29];
- "the ability to satisfy performance requirements predictably in an uncertain environment" [29].

The utility of these approaches depends to some extent on whether the uncertainty in the environment can be quantified in some way (e.g., mean time between failure statistics). If so, this information can be used by proactive scheduling techniques. If the degree of uncertainty in the environment is very high, however, or if the uncertainty is "unknown", a more reactive approach to scheduling may be appropriate.

Despite the fact that relatively little work has looked at the creation of robust schedules, predictive schedules are being executed everyday in the presence of environmental uncertainty. There exist a number of standard techniques within

manufacturing scheduling, for example, to achieve some level of robustness. These techniques include:

- Manipulation of production lead time. This technique is used in MRP/MRPII systems, where simple dispatch rules are used during schedule execution. Safety lead-time is "an element of time added to normal lead-time for the purpose of completing a job in advance of its real need date" [45]. In a make-to-order environment, the extra lead time is inserted into the schedule to absorb events such as machine breakdowns and demand surges. Such lead times are typically obtained from historical data on the manufacturing of specific orders. The amount of safety lead-time to insert is critical: too much time may result in low machine utilization, high inventory costs and uncompetitive due dates. Too short a lead-time may result in high tardiness costs. A number of techniques for determining safety lead-times are surveyed in [25].
- Use of inventory buffers both within the factory (e.g., between production stages) and at the either end of larger decompositions of the production process. For example, the use of input buffers for raw material entering a factory as well as output buffers for finished goods awaiting delivery are both common.

These approaches are expensive. Both techniques result in a significant retardation in the speed at which production expenses can be recouped from the customer. In the case of inventory buffers, further storage costs and potential safety risks (due to crowding of production facilities) are introduced. Furthermore, both long lead times and large inventory buffers inhibit the ability of the enterprise to quickly respond to external events, potentially resulting in significant lost revenue, customers, and reputation.

These techniques also act to obscure sources uncertainty. Rather than characterizing and dealing with uncertainty, enterprises use lead time and buffers to attempt to ensure that the uncertainty will have little impact. As the information technology and supply chain optimization techniques mature, this lack of insight into the uncertainty will limit the extent to which gains in efficiency can be achieved.

*1.3.2. Reactive scheduling*

Reactive scheduling takes place at the time of the execution of the schedule. Based on up-to-date information regarding the state of the system and perhaps

based on an existing predictive schedule, reactive techniques decide when and where activities should execute.

At one extreme, reactive scheduling may not be based on a predictive schedule at all: job dispatching can be done at execution time. Allocation and sequencing decisions take place dynamically in order to account for disruptions as they occur.

A less extreme approach is to completely regenerate a new, up-to-date predictive schedule when schedule breakage occurs. This approach may in principle be capable of maintaining optimal solutions, however computation times are likely to be prohibitive and production may be significantly delayed while regeneration of the schedule takes place. Furthermore, frequent schedule regeneration can result in instability and lack of continuity in detailed shop floor plans, resulting in increased costs attributable to what has been termed "shop floor nervousness" [32].

Finally, reactive scheduling techniques might simply repair the existing predictive schedule to take into account the current state of the system. Repairs may take the form of simple, fast control rules to make decisions within some real time execution constraints and to tend to minimize the perturbation to the original schedule.

In practice, it appears that the final technique is the most commonly used. Human schedulers often use their experience with the system to reassign and reroute jobs so that processing can continue. Scheduling technology that provides opportunity to perform "what-if" scenarios supports this approach, however, it is unclear how much automated reactive scheduling is done.

*1.3.3. Proactive and reactive scheduling*

A scheduling system that is able to deal with uncertainty is very likely to employ both proactive and reactive techniques. A proactive technique will typically require a, perhaps trivial, reactive component to deal with the occurrence of uncertain events during schedule execution: obviously, the schedule, exactly as defined, cannot continue to be executed with a broken machine, therefore some execution time reasoning is necessary even if it is to put in place a contingent schedule that was proactively computed. Furthermore, it is unlikely that it will be worthwhile to take into account all unexpected events proactively. Some will be too improbable or too minor and therefore if they do occur will have to be dealt with reactively. Similarly, as we discuss below, a major constraint on re-

active scheduling is the timeliness of the response. This requirement means that optimization at execution time is not a realistic goal. However, use of a proactive technique may provide the reactive component with strict bounds on the complexity of its computation while possibly allowing higher quality solutions to be found.

### *1.4. Plan of paper*

The research reviewed in this paper is organized in two ways. The first, more implicit organization concerns the placement along the proactive/reactive axis noted above. We begin with work that focuses more on the proactive techniques and then move toward more the reactive. Within this organization, we categorize the work based on more specific characteristics of its approach to scheduling with uncertainty.

Section 2 examines proactive techniques that account for uncertainty by inserting some form of redundancy, typically extra time, into the schedule. This is followed, in Section 3, by techniques using more formal probabilistic reasoning and then, in Section 4, by techniques which create multiple schedules to deal with different contingencies that may arise during schedule execution. Section 5 addresses approaches that more explicitly make use of both off-line (proactive) and on-line (reactive) scheduling while the final section, Section 6 looks at techniques for rescheduling at execution time.

While each of the above sections contains brief discussions of the work that is presented, in Section 7, we present a more general discussion of all the work that is surveyed. We conclude in Section 8.

## 2. Redundancy-based Techniques

The main characteristic of the work reviewed in this section is the reservation of extra time and/or resources so that unexpected events during execution (e.g., machine breakdowns, long activity durations) can be dealt with by using some of this "extra" time and resources.

### *2.1. Fault Tolerant Real Time Scheduling*

While there are a number of differences between scheduling problems and solution techniques typical of manufacturing and project scheduling and those

typical of the scheduling of real time systems, a critical component of providing real time guarantees for schedule execution is the ability to cope with faults. Therefore, there has been significant work done on dealing with uncertainty for real time systems.

Redundancy is the typical way in which fault tolerance is guaranteed. Two forms of redundancy are common:

1. Resource redundancy - Multiple identical sets of resources are used to execute multiple versions of each task in parallel. Error detection, which can be non-trivial in such systems, can then be provided by a voting mechanism. Resource redundancy is the only technique that can deliver real time guarantees in the presence of permanent failures (e.g., a disk crash) [17].
2. Time redundancy - Time is reserved to re-execute tasks that fail. If the primary task does not fail, the backup task then does not have to be executed.

Resource and time redundancy can be combined in a multi-processor system by scheduling backup tasks on different processors from the primary task.

For the types of scheduling problems we are interested in here, pure resource redundancy is unrealistic. The cost of providing redundant resources and/or running the same task multiple times in parallel is prohibitive. It is unrealistic that a manufacturer, for example, would double its resources and inventory production. Time redundancy, however, is relevant.

In general, there are two types of time redundancy. The first, which we will refer to as *a priori redundancy*, generates a schedule with all primary tasks and as many backup tasks as necessary to guarantee the desired level of fault tolerance. The second type of time redundancy is *a posteriori redundancy* in which the schedule is initially created without redundancy. Slack time is then inserted into the existing schedule. In both cases, the amount of redundancy that is inserted is determined by the fault model. No system can cope with an arbitrary number of faults within a time interval. Therefore, the some worst-case fault frequency must be assumed. For example, Ghosh et al. [18] assume a queue-based schedule, where the sequence of tasks on the resource is already generated. Based on an analysis of the fault frequency and worst-case activity duration, back-up tasks are inserted into the queue. Rather than corresponding to specific primary tasks, these back-up tasks simply reserve time for re-execution in the event of a fault. It is not always the case that sufficient redundancy can be inserted to guarantee fault tolerance and the meeting of all due dates.

The work in real time systems has developed a rich mathematical basis for the determination of the slack time that must be added to a schedule. Unfortunately, much of this formalism depends on the theoretical characteristics of the scheduling problem and the techniques for solving the scheduling problems. While it is hoped that this mathematical foundation is extendible to the scheduling problems we are interested in, one challenge will arise from the difficulty of the underlying scheduling problem as well as the variety of scheduling approaches. With multiple resources and complicated interrelations among activities, standard models of scheduling in AI and OR are usually NP-hard. The pre-emptive single processor scheduling model often used in real time scheduling has polynomial solutions. This difference in complexity for the underlying problem may make the adaptation of the fault tolerance guarantee challenging.

### *2.2. Slack-based Protection*

Leon et al. [31] approach robust scheduling by redefining the evaluation function of a schedule to include an expression of robustness. Given such an evaluation function, optimal schedules can be found using traditional OR search techniques.

In order to define an evaluation function, a number of robustness measures are developed and evaluated. The problem model used is as follows: let $S$ be a job shop schedule specifying the order in which activities are executed on machines, and let $M_o(S)$ be the deterministic makespan of $S$ assuming no disruptions. Let the random variable $M(S)$ denote the actual makespan of $S$ in the presence of disruptions. No activity can be processed during a disruption and disrupted activities must be restarted from the beginning. When disruptions occur, schedule breakage is fixed by applying the Right Shift rule, described in section 6.1. This rule pushes activities affected by the disruption forward in time, without changing the sequence of activities processed by any machine.

The schedule delay is defined as a random variable expressing the difference between executed and deterministic schedule makespan:

$$\delta(S) = M(S) - M_o(S) \tag{1}$$

Since $M_o(S)$ is deterministic, we can write the expected values of $M$ and $\delta$ as:

$$E[M(S)] = E[\delta(S)] + M_o(S) \tag{2}$$

$$E[\delta(S)] = E[M(S)] - M_o(S) \tag{3}$$

A small value for expected delay implies that the schedule is affected little by random disruptions. However, zero delay can be achieved by inserting large amounts of idle time into the schedule, therefore simply minimizing expected delay is unlikely to lead to usable schedules. Expected makespan is also important in maximizing the use of the resources in the scheduling environment, therefore, the authors define schedule robustness as a linear combination of expected makespan and delay. Let $r$ be a real valued weight in the interval $[0, 1]$. Schedule robustness, $R(S)$, is defined as:

$$R(S) = r \times E[M(S)] + (1 - r) \times E[\delta(S)] \tag{4}$$

The authors demonstrate that $R(S)$ can be computed directly for a schedule when only one disruption occurs. However, when there is more than one disruption, exact calculation is intractable since the effect of one disruption depends upon the outcome of all previous disruptions. For such problems the authors develop three surrogate measures of robustness based on the deterministic makespan and some value correlated with expected delay. The simplest such measure, $RM3(S)$, rests on the assumption that the expected delay is negatively correlated with the average activity slack.

The slack for activity $A_i$, $slack_i$, is the amount of room $A_i$ has to shift within the schedule without breaking any constraint nor extending the makespan. Slack is simply the number of possible start times that an activity has within a schedule as shown in equation 5.

$$slack_i = lst_i - est_i \tag{5}$$

There is a negative correlation between activity slack and delay in the schedule because the amount of slack represents the idle time that can be used, in some cases, to absorb unexpected events. Therefore, as one surrogate measure of robustness, Leon et al. suggest $RM3(S)$, the deterministic makespan minus the mean activity slack, as shown in equation 6:

$$RM3(S) = M_o(S) - \frac{\sum_{i \in N_f} slack_i}{|N_f|} \tag{6}$$

where $N_f$ is the set of activities executing on fallible machines.

The surrogate measures were evaluated using a simulation study. Interestingly, the simulation demonstrated that the mean activity slack was as good a predictor of $E[\delta(S)]$ as the more sophisticated surrogates. Furthermore the $RM3(S)$ was found to perform better than the exact calculation of expected delay for the single disruption case.

In a subsequent experiment it was shown that using an optimization criteria that was a linear combination $RM3(S)$ and mean activity slack enabled search to find schedules with a smaller expected delay and a only slightly longer expected makespan than when makespan minimization was the criteria.

There are a number of comments to be made about this work:

- The $R(S)$ robustness measure is one of the few attempts outside the real-time scheduling community to formalize the definition of schedule robustness. Unfortunately, the measure conflates the notion of robustness with the traditional job shop optimization criteria of makespan minimization. A formalization of schedule robustness must be independent of a specific optimization criteria if it is to be useful. It is likely to be necessary to balance robustness against other measures of schedule quality and therefore including a particular optimization criteria in the definition of robustness limits its applicability.
- This work is considered in the context of the Right Shift control rule for recovering from machine failure. More sophisticated rules exist [39] and, indeed, complete rescheduling can be viewed as an extreme form of recovery (see Section 6.1). However since the more sophisticated rules do not preserve activity orderings on a resource, they are even less amenable to mathematical analysis than even Right Shift rule.

### *2.3. Temporal Protection*

Another redundancy-based technique for schedule robustness is the use of *temporal protection* [7,15] which extends activity durations to account for the possibility of machine breakdown. Since the actual execution time will be shorter, the extra time is used in the event of schedule disruption.

A temporally protected activity is composed of an inner interval $P_{inner}$ and an outer interval $P_{outer}$ sharing the same end time and with `lower-slack` being the difference as shown in Figure 1.

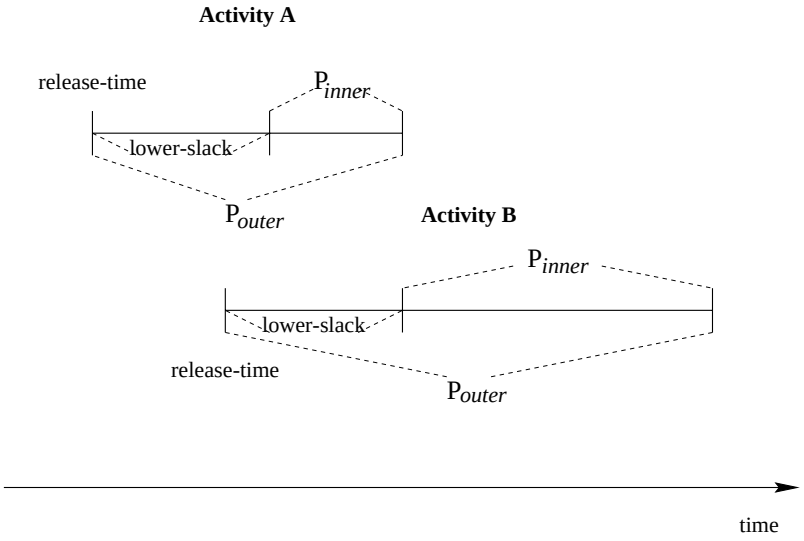


Figure 1. Temporal protection, illustrating overlapping activities $A$ and $B$, where $A$ must execute before $B$

$P_{inner}$ defines the start time and the time during for which the activity is expected to execute and the interval over which critical (that is, fallible) resources are allocated. $P_{outer}$ defines the earliest time the activity is expected to be able to start and the interval over which non-critical resources are allocated. When executing a schedule, if the previous activity takes less time than the protected duration, the critical resources are released early and so the activity can start before $P_{inner}$.

$P_{inner}$ and $P_{outer}$ are functions of the original activity duration and resource failure statistics. Their actual lengths are obviously an important issue in balancing robustness against other schedule performance metrics.

The following parameters were used in formulating the amount of temporal protection:

- $P$, the original processing time of an activity;
- $F$, a random variable representing time between machine failure;
- $D$, a random variable representing the duration of a failure.

$P/F$, therefore, is the expected number of breakdowns during the processing of an activity. The extended duration $P_{ext}$ with machine breakdowns is:

$$P_{ext} = P + \frac{P}{F} \times D \tag{7}$$

If the mean of $D$ and $F$ are known, as $\bar{D}$ and $\bar{F}$, then we can calculate the mean of $P_{ext}$ as follows:

$$P_{mean} = P + \frac{P}{\bar{F}} \times \bar{D} \tag{8}$$

Instead of being random variables of known distribution, the failure duration and time between failures may only be known to be bounded approximately. Let the bounds be $(D_{lb}, D_{ub})$ for $D$ and $(F_{lb}, F_{ub})$ for $F$. We can then calculate the lower and upper bounds on $P_{ext}$ as follows:

$$P_{extlb} = P + \frac{P}{F_{ub}} \times D_{lb} \tag{9}$$

$$P_{extub} = P + \frac{P}{F_{lb}} \times D_{ub} \tag{10}$$

Four different combinations of statistics were used for the calculation of the temporal protection. A preliminary single-machine scheduling experiment indicated that the most robust performance was achieved with $P_{inner}$ set to $P_{mean}$ and $P_{outer}$ set to $P_{extub}$.

Experiments were then run on a set of job shop problems with simulated schedule execution. Results indicated that temporal protection significantly reduced the deviation between predicted and executed makespan and that, while work-in-process cost was reduced over non-protected schedules, the tardiness costs of protected schedules were higher.

### *2.4. Discussion*

For tractable classes of scheduling problems, such as those studied in the real-time fault tolerant systems work, there has been significant formalization of redundancy-based techniques for schedule robustness. While it is unclear at the moment whether the formality of these techniques can be usefully extended to harder scheduling problems, the work presents a firm foundation.

The work that has been done with more complex problem classes seems preliminary but does, however, provide some insight. In particular, it seems that the more global measure of average activity slack has distinct advantages over temporal protection. In the latter, the durations of each activity must be extended. At execution time if no breakdown occurs it is difficult for the "extra" time that was part of the extended duration to be shared with subsequent activities on the critical resource. Reasoning about slack time, on the other hand, allows such sharing as the slack between consecutive activities can be used by either depending on the timing of the machine breakdowns.

Unfortunately, with both techniques, the critical issue is the amount of slack (or duration extension) that should be used given the uncertainty conditions of the schedule.

A number of other points about this work should be noted:

- While the real-time fault tolerant community has looked at hardware redundancy for dealing with non-transient faults, there is no work that we are aware of that uses resource redundancy in more complicated classes of scheduling problems. However, in situations where switching among resources is not too costly, it might be useful to reserve extra capacity to deal with uncertainty.
- The temporal redundancy work treats all activities on critical resources equally. In fact, however, there is a much smaller likelihood of resource breakage before or during the execution of an activity early in the schedule than late in a schedule. Therefore, it would seem useful to bias any method for dealing with uncertainty to have an increasing effect across the scheduling horizon.
- Given the insight of Leon et al., that mean slack time of each activity was a good predictor of delay, we can imagine a number of ways to control the slack time of activities. For example, we could directly constrain each activity to have at least some level of slack (perhaps based on the earliest time at which it can be scheduled). Alternatively, we could insert breaks on the critical resource during the original scheduling. These breaks can be removed at execution/rescheduling time, resulting in slack that can be used to deal with disruptions. The problem, as we noted above, is still deciding on how much slack should be added to the schedule.
- Finally, it is interesting to note that the work on redundancy has really only addressed uncertainty arising from machine breakdowns. While temporal redundancy may well work with other sources of uncertainty, it is even less clear

how to determine the appropriate level of redundancy under conditions where, for example, activity durations are variable, orders may change or deliveries of raw materials could be late (or early).

## 3. Probabilistic Techniques

Redundancy-based techniques address the problem of uncertainty with the assumption that adding redundancy can be a solution. Probabilistic techniques take a different approach. Rather than starting with the assumption that redundancy is the solution, they start from the position that a key piece of information is the probability that the schedule can be executed. *A priori*, this is a diagnostic tool rather than a solution: it does not produce robust schedules, rather it allows the uncertainty in a schedule to be measured. However, once we have the ability to measure such probabilities, we may also be able to build schedules so as to maximize them. For example, perhaps we use probabilistic techniques to guide the amount and location of redundancy that should be inserted into the schedule?

### *3.1. Probabilistic Real Time Fault Tolerant Scheduling*

The standard form for real-time system fault tolerance guarantees is to assume a fault model as part of the problem definition. A schedule is created based on the fault model and the typical fault tolerant guarantee is that all due dates will be achieved as long as the faults arrive no more quickly than assumed by the fault model.

Burns et al. [4] introduce the notion of a *probabilistic guarantee* for hard real-time systems. This probabilistic guarantee is a guarantee of schedulability with an associated probability. In particular, this means that a guarantee of 99% for a schedule does not indicate that 99% of the jobs will meet their due dates but rather that in 99% of the executions of this schedule, all jobs will meet their due dates. This is a similar notion of probabilistic customer service levels in inventory management where the level of inventory allocated to a warehouse is such that all customer orders will be met some percentage of the time.

Burns et al.[2] use a single processor, pre-emptive, fixed priority scheduling model. In such a model a finite number of tasks must be repeatedly executed. Each task, $\tau_i$, has a minimum inter-arrival time, $T_i$, a worst-case execution time,

[2] Unless otherwise noted, the details in this section are taken from [4].

$C_i$, and a deadline, $D_i$, (relative to its arrival) where the deadline of each task is less than or equal to the minimum inter-arrival time. Tasks may be pre-empted at any time during their execution. The problem, then, is to define a policy which guarantees that each instance of each task will finish by its deadline. Such a policy, if it exists, can be found by assigning a priority, $P_i$, to each task, such that $D_i < D_j \rightarrow P_i > P_j$, and by always executing the highest priority task. This execution includes pre-empting a task if a higher priority task arrives.

To incorporate faults, Burns et al. define $F_i$ to be the extra time needed by task $\tau_i$ if a fault occurs during its execution. Depending on the scheduling model, $F_i$ may represent the duration of the fault plus:

- the amount of time left for $\tau_i$ to execute after the fault,
- the time required to completely re-execute $\tau_i$ after the fault, or
- the time required for some "recovery" operation followed by either of the two above amounts.

Regardless of the actual semantics of $F_i$, the worst-case completion time of each task, $R_i$, can be obtained from solving the recurrence relation (adapted from [4]):

$$r_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j + \left\lceil \frac{r_i^n}{T_f} \right\rceil \max_{k \in hp(i) + \{\tau_i\}} F_k \tag{11}$$

Where:

- $hp(i)$ is the set of tasks with higher priority than task $\tau_i$.
- $T_f$ is the minimal inter-arrival time for faults.
- $r_i^0 = C_i$.

The summation term is the delay during the execution of $\tau_i$ resulting from pre-emption by higher priority tasks while the final term is the delay due to the occurrence of faults. The authors note that faults are essentially treated as sporadic tasks with priority higher than any other task.

When $r_i^{n+1}$ becomes equal to $r_i^n$, this value is also equal to $R_i$. However, it is possible that $r_i^n$ becomes greater than $D_i$, in which case the system is unschedulable as no deadline guarantee can be given.

Instead of providing a firm guarantee, however, the authors use sensitivity analysis to find $T_F$ which is the minimum value of $T_f$ such that all tasks meet

their deadlines. A probabilistic guarantee can then be provided by finding the probability that the observed minimum inter-arrival time for faults, $W$, is less than $T_F$.

If $Pr(U)$ is the probability that a system is unschedulable and the conservative assumption is made that any interval between faults that is less than $T_F$ will result in an unschedulable system, then the probability that the system is unschedulable is equal to the probability that the minimal arrival time between faults is less than $T_F$: $Pr(U) = Pr(W < T_F)$. The formulation of $Pr(W < T_F)$ requires use of the Taylor series approximation and is beyond the scope of this review.

While no empirical results are provided, the authors demonstrate that their Taylor series approximation achieves a very high degree of numerical accuracy.

### *3.2. $\beta$-Robust Scheduling*

Rather than faults as the source of uncertainty, another model is that of duration or processing time uncertainty. That is, the time that each activity must execute is not precisely known and the goal is to produce a schedule with the maximum probability of achieving a specific level of some performance measure.

Daniels & Carrillo [10] introduce the notion of a $\beta$-robust schedule for a single-machine scheduling model with processing time uncertainty. In particular, the authors note that with such a source of uncertainty, it is insufficient to simply consider the mean value of a measure of schedule quality (e.g., mean flow time) as the variance provides critical information. Under uncertainty, a schedule with optimal mean performance may have an extremely high variance while a schedule with a sub-optimal mean performance may have a much lower variance. If it is important to minimize the risk of unacceptable performance rather than achieve optimal performance, the latter schedule may be preferable.

Clearly, the expected system performance depends both on the actual processing time of each job and on the sequence of jobs on the (single) machine. Given a desired level of system performance, $T$, a $\beta$-robust schedule is the sequence of jobs that maximizes the probability of achieving system performance less than or equal to $T$. This notion is formalized by considering $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{|\Lambda|}\}$, where $\Lambda$ is the ordered set of all scenarios, $\lambda_i$, such that each scenario assigns a processing time to each job. The ordering of $\Lambda$ is in descending order of system performance given a particular sequence of jobs, $\pi$.

Clearly, the system performance and therefore the ordering of the scenarios in $\Lambda$ can be different for each sequence of jobs.

Given a job sequence and the ordering of scenarios, it is easy to identify $\lambda_{j*} \in \Lambda$, the scenario with highest rank in the order that has a performance value of less than or equal to $T$.

Assembling these components, the authors form equation 12 which expresses that the the $\beta$ robustness value of a sequence of jobs can be found by summing the probabilities of each scenario under which the desired level of system performance is achieved.

$$\beta(\pi, T) = \sum_{j=1}^{j*} P(\lambda_j) \tag{12}$$

Where $P(\lambda_j)$ is the probability that the actual processing time of each job follows scenario $\lambda_j$.

The final step is, then, to find the "$\beta$-robust schedule", the sequence, $\pi_\beta$ that maximizes $\beta(\pi_\beta, T)$.

Assuming that the processing time of each job is an independent random variable with a known mean and variance and that the performance measure is the flow time, Daniels & Carrillo reformulate equation 12 and prove that the resulting problem is NP-hard. Note that this complexity results arises even though the underlying scheduling problem (without uncertainty) can be solved in $O(n \log n)$.

A branch-and-bound solution procedure including dominance rules, bounding techniques, and heuristics for branch selection is developed for this problem. Empirical results using expected flow time as the performance measure demonstrate that the $\beta$-robust schedule significantly reduces the risk associated with execution of the schedule as compared with the shortest expected processing time (SEPT) dispatch rule which does not take into account uncertainty: the $\beta$-robust schedule resulted in an mean error of 0.2% above the optimal expected flow time whereas the SEPT rule resulted in mean error of 14.1%.

Further experiments examined an extended problem in which there is the opportunity of allocating a resource to a job in order to reduce its processing time variation. The authors indicate that such a model could be used, for example, in situations where management time can be allocated to interviewing customers so as to achieve a better understanding of their detailed requirements. In such a situation, a $\beta$-robust schedule must not only find the optimal sequence of jobs, but

also the optimal allocation of resources to jobs so as to maximize the probability of acceptable system performance. Again, $\beta$-robust schedules were shown to achieve significantly higher system performance even when compared to SEPT schedules to which the optimal resource allocation was applied.

### *3.3. Multiobjective Stochastic Dominance A* Search*

In recent years much work has been carried out in the AI planning community on *decision theoretic planning*. This differs from classical AI planning in the following ways [46]:

- the effects of actions are described by probability distributions over outcomes;
- objectives are described by utility functions;
- the criteria for effective plan generation is expected utility maximization.

Although this work seems applicable to scheduling, and a number of researchers have suggested using decision theory to deal with uncertainty in the scheduling domain [29,11], little work has been carried out in this area. Here, we look at work that applies multiobjective stochastic dominance A* search on a single-machine scheduling problem. In Sections 4.2 and 5.4 we look at other work within decision theoretic planning with contingent and off-line/on-line approaches to uncertainty, respectively.

Wurman & Wellman [48] address a single machine, stochastic processing time, sequence dependent stochastic setup time scheduling problem with due dates from the perspective of state space search. The optimization criteria is the expected weighted number of tardy jobs.

The actual problem studied is the stochastic lot-sizing problem which consists of a set of orders for different inventories and a task schema defining production and shipping tasks for different quantities of each inventory and for setting up the machine to produce each inventory. The processing time of each production task is stochastic but proportional to the amount of inventory produced. Shipping tasks are instantaneous and do not require the machine while setup tasks have a stochastic processing time independent of the preceding and succeeding production tasks. Each order has a due date and a penalty that must be paid if the inventory is not shipped by the due date.

The authors adopt a state encoding that consists of a set of orders (and whether they have already been met), the quantity of each inventory that cur-

rently exists, and the current machine setup. The cost of a state is represented by a pair of distributions: one for the accrued cost and one for the time.

The authors show that this state encoding requires the use of Multiobjective Stochastic Dominance A* (MO-SDA*) to ensure a sound and complete search of the state space. The precise definition of MO-SDA* is beyond the scope of this paper, however, it should be noted that the critical contribution of the method is a necessary and sufficient basis on which paths in the search can be pruned.

A careful definition of the conditions in which particular operators can be applied is done to significantly limit the size of the search space (but yet not remove optimal solutions). Finally, two heuristics are defined, based on problem relaxations, to aid in the evaluation of promising nodes.

Empirical results compared the MO-SDA* solutions with the solutions found for the deterministic model (e.g., with the stochasticity ignored). Execution of the two schedules is then simulated (under the stochastic conditions) and the overall cost of the schedules was compared. Based on a set of 700 randomly produced problems (with total estimated capacity ranging from 95% to 125% and ranging from 6 to 20 orders), results indicated that the stochastic schedule always achieved cost at least as small as the deterministic schedule. However, in the problems above 100% capacity, the stochastic schedule was strictly superior in almost three-quarters of the problems. Comparison of search effort shows that in problems with 100% or higher capacity, the MO-SDA* expands approximately twice as many nodes as the deterministic algorithm.

### *3.4. Discussion*

There are a number of interesting issues raised by a comparison of the work reviewed in this section with the redundancy based work. First, it would appear that there is a subtle difference in the problems that are actually being solved. The probabilistic work attempts to find a solution which maximizes some reward function based on the meeting of due dates for each order. The operators available are simply the usual scheduling operators (i.e., the sequencing of activities in the examples above). In contrast, the redundancy based techniques change the actual problem definition by inserting slack requirements into the original problem. A good redundancy-based solution allows the user visibility into the actual time at which each activity will complete. In contrast, a probabilistic based solution communicates the information about the probability that the original due date

of the job will be met. Ideally, in terms of coordination with other supply chain components, we would like a mix of information: a completion time together with a probability that that will be the true completion time. Obviously, it is important that the schedule attempts to make this completion time close to the original due date, however, it is often more important to establish a completion time with a high probability of achieving it. This is especially the case in situations with the opportunity for negotiation with the customer and where there is a complex supply chain of which the schedule is only one part.

The second point of comparison concerns the relationship between redundancy and the probabilistic analysis. There would seem to be a close relationship between the ability to insert time redundancy in order to guarantee schedulability (under an assumed fault model) and the probabilistic level of schedulability: that is, if it is possible to insert slack to guarantee fault tolerance, the probabilistic analysis is also able to establish that the system is 100% schedulable without explicit slack inserted. Furthermore, in situations where a guarantee is impossible, the probabilistic analysis provides more information: a probability that the system will experience failure due to faults. This suggests that explicit insertion of slack time may not be necessary. Rather, combined with the point made just above, it is sufficient to simply manipulate the completion time of the jobs: minimize tardiness under the requirement that each order must have some level of probability of achieving its completion date. A critical point, then, is the tractability of reasoning about the probability that each order will achieve its completion date, especially with more complex underlying scheduling models such as JSSP. In such case, it may be that explicit insertion of redundancy is necessary, perhaps as an approximation, in order to allow practically sized problems to be solved.

The MO-SDA* work of Wurman & Wellman shares the advantages of direct probabilistic reasoning just noted. However, given its origin as an AI planning technique, it is explicitly based on state-space search. Such search techniques can be applied to problems that are significantly more general that scheduling problems. Given the specialized scheduling techniques developed in the AI and OR fields, it is an open question whether the state-space search infrastructure is necessary even for scheduling problems including uncertainty.

It seems reasonable that explicit probabilistic reasoning is necessary to achieving formality in dealing with uncertainty in scheduling. Indeed, the redundancy based techniques often derive the redundancy inserted in the schedule

based on some, perhaps informal, probabilistic reasoning. The critical issue that arises from the probabilistic work reviewed here is the tractability of the reasoning versus the benefit. All the examples here apply probabilistic reasoning to a one-machine scheduling problem resulting in a problem that is NP-hard, even though the underlying problem is polynomial. Adapting these problem models to harder scheduling problems (e.g., JSSP or RCPSP) which are already NP-hard, would seem even more challenging.[3] Therefore, while extending such models to harder scheduling problems is necessary from a theoretical perspective, it would appear that approximation techniques and heuristics employing some probabilistic reasoning is a more viable approach to these problems in practice.

## 4. Contingent Scheduling

Redundancy and probabilistic techniques are based on the approach of generating a single schedule that is likely to be able to incorporate unexpected events without major disruptions. A different approach is represented by contingent scheduling techniques. These techniques are based on attempting to anticipate likely disruptive events and generating multiple schedules (or schedule fragments) which optimally respond to the anticipated events. This is all done *a priori* so that at execution time a set of schedules are available. Responding to unexpected (but anticipated) events and execution time simply consists of switching to the schedule that corresponds to the events that have occurred.

### *4.1. Just-in-Case Scheduling*

Just-in-Case (JIC) scheduling [11] is a technique for generating schedules in a domain where activities have uncertain durations, which can lead to schedule breakage. It has been developed and applied in the domain of telescope observation scheduling.

The telescope observation problem consists of a single resource, the telescope. Each job is of an observation activity, which has a time window determined by the possible observation times for the activity. The activity durations are stochastic, modeled using a normal distribution using statistics from previous execution data.

[3] However, as Wurman & Wellman note, some NP-hard scheduling problems become polynomial (under strict assumptions) when uncertainty is added.

One way of dealing with duration uncertainty is to always assume the worst case: set all activity durations to their longest time possible and solve the resulting deterministic duration scheduling problem. When some activity finishes early, introduce a "wait" activity to fill up the remaining time. This technique is quite similar to the temporal protection technique discussed above (Section 2.3). A weakness of this technique is that resource usage may be very low as the duration is a worst-case estimate and therefore it is likely that the resources will be idle for much of the time. An alternative is to initially schedule with mean durations, using the Right Shift Rule (see Section 6.1) to reassign the start time of activities when some activity takes longer than its mean duration to execute. If a breakage occurs, that is, if any activity is right shifted so that it can no longer execute given its original time window, then rescheduling is performed. This approach also results in idle time during the rescheduling. The goal then is to avoid schedule breakage without sacrificing schedule quality.

Given an initial schedule and a model of how action durations can vary, JIC approaches the problem by identifying the time point where a schedule break is most likely. The break point is split into two cases: one where the schedule breaks and the other where it does not. The scheduler is then invoked on the subproblem in which the break occurs in order to generate a new schedule from that point on. This produces a schedule for a single break. Given more time, JIC can be repeatedly applied to consider more possible break cases using the multiply contingent schedule as input.

JIC scheduling has been evaluated on real telescope scheduling data [11]. One experiment showed that the amount of the schedule that could be successfully executed (without the need to reschedule on-line) increased from 62% to 96% when 10 breaks were considered. Another experiment showed that using the original, non-contingent schedule, the percentage of the schedule executed decreases rapidly with increasing duration uncertainty. As the contingency of the schedule is increased there is an increase in the percentage of the schedule executed; however, the size of the increase diminishes as the uncertainty increases, regardless of number of breaks considered. Furthermore the amount of improvement in schedule robustness diminishes as contingency increases. The maximum uncertainty considered in the experiments was characterized by the standard deviation of the duration being 10% of the mean duration. At this point, about 40% of the schedule was executed before breaking with a non-contingent schedule. With 16 breaks considered, just over 70% of the schedule was executed. Given

the diminishing returns in schedule robustness as the number of cases considered by JIC is increased, there will probably not be much improvement on this 70% figure by considering more breaks.

### *4.2. Markov Decision Processes*

Markov Decision Processes (MDPs) are a traditional tools in the AI and OR communities for decision theoretic planning (see Section 3.3). Briefly, the MDP framework consists of a number of components [3]:

- a set of states, $S$.
- a set of actions, $A$, which are defined as a probability distribution for mapping between states. A transition matrix is defined for each action, $a$, specifying the probability that action $a$ performed in state $s$ will move the system to state $s'$.
- a set of observations, $O$, which represent execution time information about the state of the system. Because there is no assumption that the agent will necessarily be able to determine the exact state of the system, a probability distribution is defined over the possible observations. For each possible observation, $o_m$, for each action, $a_k$, and for every pair of states, $s, s'$, the probability of obtaining observation $o_m$ given that action $a_k$ has moved the system from state $s$ to state $s'$ is defined.
- a value function, $V$, which maps the set of states that have been visited (e.g., the state trajectory or state history) into a real number.
- a horizon, $T$, which defines that length of the state trajectory to be used in evaluating $V$.

The goal, in the classical MDP formulation, is to find a *policy*, a mapping of state trajectories into actions, that maximizes the objective function. The objective function can vary but is typically based on the expected value of $V$ (e.g., expected total reward). See Boutilier et al. [3] for an introduction to MDPs and survey of extensions and solution techniques.

Meuleau et al. [33] apply MDPs to a resource allocation problem using the example of air campaign planning. Given a set of targets to be destroyed and a set of weapons to be allocated to targets, the goal is to compute the optimal allocation under the conditions that there is:

- a (constant) cost for each weapon used

- a (variable) reward for each target destroyed depending on the target
- a non-zero probability that the allocation of a weapon to a target will not result in its destruction
- a global limit on the number of weapons available
- a local limit on the number of weapons that can be used at each state
- a time window within which each target is vulnerable to a weapon

The local limit on weapons in each state is expressed using the constraints that there are a limited number of planes that can be used in each state and that each plane carries a limited number of weapons. In one state, a plane allocates all its weapons to a single target.

The authors conclude that the limit of tractability in applying standard MDP solution techniques to the problem is when there are 6 targets and 60 weapons: computation time was on the order of 6 hours. As the actual problems that were to be address were significantly larger (e.g., hundreds of targets, thousands of weapons, and tens of planes), the authors turned to other solution methods that are discussed below (Section 5.4).

### *4.3. Discussion*

There are two major weaknesses concerning the use of contingent scheduling to cope with uncertainty. The first issue is that of the combinatorics of contingent scheduling. The JIC approach was used in problems with one resource and considered fewer than 20 disruptions for which a contingency had to be found. Adapting JIC scheduling to a multi-resource problem appears problematic as the contingencies on different machines are combinatoric. Given ten resources which may experience breakdowns, a fully contingent schedule that accounts for one breakdown for each resource requires the creation of 1024 schedules: each resource can either break or not, independently of all other resources. Furthermore, the creation of a contingency for one break for one resource also includes specification of the time at which the breakdown occurs. These are the combinatorics with which the MDP model of Meuleau et al. has to cope. As shown in their experimental results, this strictly limits the size of problems that can be realistically addressed.

The second weakness of the contingency approach arises from the view that a schedule is likely to be part of a larger supply chain. Contingent scheduling does

not provide visibility into when jobs are actually likely to be completed. There is an initial schedule and then a set of contingencies. These contingencies can be wildly different from the original schedule and from each other. If only the original schedule is communicated to other supply chain nodes, these nodes will make their schedule based on a schedule that does not make any attempt at robustness. Unfortunately, even if the entire multi-contingent schedule is communicated, the other nodes have poor information: a job may finish at one of a set of possible times (perhaps with an associated probability) and it may finish at some time not in this set (hopefully with a very low probability). A contingent schedule becomes a major source of uncertainty when input to another node in the supply chain.

## 5. Off-line/On-Line Approaches

The techniques discussed to this point have focussed on the generation of predictive schedules that, in some way, take into account the uncertainty in the environment. While some of them (e.g., the redundancy-based and the contingency techniques) assume that some reasoning will have to be done at the time of schedule execution (e.g., shifting activities to take advantage of the slack time that was reserved or choosing one of the contingencies), in general they are off-line techniques, minimizing the need for and complexity of on-line reasoning.

In this section, we turn to work that has looked more explicitly at off-line/on-line algorithms. Traditionally, such algorithms consist of two-phases:

1. The off-line, proactive, phase which is performed sometime before the schedule has to be executed. The off-line phase typically has the luxury of significant time in which to search for a solution that will optimize or come close to optimizing schedule quality. As we will see, in some cases, the off-line phase does not actually solve the overall problem, but rather a set of sub-problems that then must be integrated.
2. The on-line, reactive, phase which is performed at schedule execution time. This phase has up-to-date information about the state of the scheduling problem but little time in which complex search procedures can be executed.

The work reviewed in this section does not necessarily explicitly discuss both off-line and on-line techniques but rather is written from the perspective that both phases are necessary even if only one is discussed. For example, the

work on least and delayed commitment scheduling and the work on supermodels, both focus on off-line techniques without explicitly detailing an accompanying on-line technique. This is because a variety of on-line techniques are possible and that the theme of the work is, in the former case, to make decisions only when the appropriate level of information is available, and, in the latter case, to solve the off-line problem so that strong theoretical limits can be placed on the amount of work required by an on-line phase. The other two pieces of work discussed in this section present both off-line and on-line phases from the perspective of artificial immune systems and from the perspective of decomposed Markov Decision Processes.

### *5.1. Least Commitment and Delayed Commitment Scheduling*

A common approach to off-line algorithms that specifically assume, but do not necessarily define, an on-line counterpart is least or delayed commitment scheduling. In general, the two terms refer to the creation of a predictive schedule that does not completely define all characteristics of all activities to be executed. Rather, a set of constraints are added to the scheduling problem significantly narrowing the search space that needs to be explored in an on-line phase. Least and delayed commitment [2] are based on the idea that decisions should not be taken where information is incomplete or uncertain if it can be avoided. In highly uncertain environments, it may be better to only generate predictive schedules for a short time in the future, since they are highly likely to break soon anyway. On the other hand, in more certain environments it may be useful to vary the level of commitment across the scheduling horizon. In the near term, generate schedules with a high level of commitment, but further in the future make weaker commitments which can be refined when more information about the state of the world is known. This idea is explored in [37], and more recently in the sliding scale of commitment approach [26,27].

For example, a least/delayed commitment approach to the JSSP is to post sequencing constraints between the activities on each resource, rather than assigning specific start times. A single sequencing solution represents many possible start time solutions. The actual start times of the activities can be found with a polynomial technique in the on-line phase which may take into account preferences and up-to-date operating conditions on the shop floor. The sequence solution can absorb minor variations in schedule execution; for instance an activ-

ity starting later than planned as a result of an unplanned disruption may still be able satisfy the sequencing constraints posted in the scheduling solution.

Though we do not make a distinction between least and delayed commitment scheduling in this paper, there is often a subtle difference between the two approaches. Least commitment scheduling typically provides a guarantee that at least one solution exists in the search space defined by the predictive schedule while delayed commitment scheduling provides no such guarantee.

In addition to the work we have already noted, least and delayed commitment approaches have been adopted by a number of scheduling projects, including:

- the Honeywell Batch Scheduler [20], where it is called constraint envelope scheduling
- the Dynamic Scheduling System [24,35]
- the ODO scheduling project [1]
- work associated with Carnegie Mellon University [6,5]

A weakness of many of these projects is that while robustness is seen as a goal of the least and delayed commitment scheduling, little is done to define robustness and there tends to be no experimental work in determining if the approach actually leads to more robust schedules. The work typically reported focuses on effort to find least commitment predictive schedules. One exception is the work of Cesta et al. [5] where a robustness measure is formally defined and schedules are generated that are, partially, rated by this measure. Again, however, the focus of the work is the solving methods and so no evaluation of the actual (or simulated) robustness of the solutions is performed.

An interesting least/delayed commitment approach together with simulation of schedule execution is explored in Wu et al. [47]. This work attempts to identify some critical subset of scheduling decisions that, to a large extent, dictate global scheduling performance. The scheduling problem is solved for these decisions while the remaining decisions are left to be resolved later. The critical decisions which are made result in a sufficiently detailed "sketch" of the schedule to serve as a basis for other system planning requirements, yet retain enough flexibility such that disturbances and detailed shop constraints can be dealt with dynamically during execution.

The problem studied is job shop scheduling to minimize weighted tardiness. The least commitment approach works by only making some of the sequencing

decisions necessary for a full solution to the scheduling problem. The activities are decomposed into a sequence of mutually exclusive, exhaustive subsets, specified by an *ordered assignment*. The order of the subsets establishes additional precedence relationships between the activities. For instance, if activity $A$ is assigned to subset $e_s$ and activity $B$ is assigned to $e_t$, and $e_s$ precedes $e_t$, then there exists a precedence constraint between $A$ and $B$.

For each assignment of activities to subsets, we can associate a cost function which is the minimum weighted tardiness with the extra precedence constraints induced by the assignment. The problem we are now concerned is the *ordered assignment problem* (OAP) where the goal is to partition the activities into ordered subsets such as to respect the precedence constraints that are already specified and to minimize the cost function.

Wu et al. develop a branch and bound scheme to generate schedules by solving the OAP followed by partial scheduling of the subsets. Experimentation used an on-line algorithm based on a dynamic dispatching heuristic that respects the precedence constraints specified by the OAP solution. These experiments showed that solving the OAP before partial scheduling significantly improved scheduling performance and provided more robust performance under a wide range of disturbances than traditional static and dynamic scheduling methods.

### *5.2. Supermodels*

Though not directly applied to scheduling, Ginsberg et al. [19] present the *supermodel* concept to characterize robust solutions. This work is motivated by the need to build off-line solutions that can be quickly repaired on-line with a small set of modifications.

The work is presented in the context of the SATISFIABILITY problem [16] in which a solution is called a model. A supermodel is defined as follows:

> **Definition 4:** An $(a, b)$-supermodel is a model such that if we modify the values taken by the variables in a set of size at most $a$, another model can be obtained by modifying the values of the variables in a disjoint set of size at most $b$.

For example, a $(1, 1)$-supermodel guarantees that if the value of any single variable is changed, we can recover a model by changing the value of at most one other variable.

A proof, demonstrating that if finding models is in NP then finding supermodels is also in NP, is used as a basis for the generation of supermodels. Rather than rely on a problem solver which knows about and attempts to generate supermodels, the original problem $P$ is transformed to a new problem $P'$ by adding extra variables and clauses which specify that any solution to $P'$ is an $(a, b)$-supermodel of $P$. The advantage of this technique is that any problem solver for $P$ can also be used to generate supermodels.

For random 3-SAT problems it is well-known that a phase transition between solvable and unsolvable instances is found where the ratio of the number of clauses to the number of variables is about 4.2 [9]. When this ratio is higher, problems tend to be overconstrained and unsatisfiable. Below this ratio problems are underconstrained and almost always satisfiable. The mean cost of finding a solution or proving unsatisfiability tends to be highest at this ratio. Experimental results show that the phase transition peak for finding $(1, 1)$-supermodels of random 3-SAT occurs around a ratio of 1.5, which is in the relatively underconstrained region for 3-SAT. $(1, 1)$-supermodels were not found for more highly constrained problems.

These results indicates that the full supermodel concept is applicable only to very underconstrained problems. More constrained problems, though they may have many models, do not appear to have any $(1, 1)$-supermodels. While they may have higher order supermodels, the modification of the underlying problem to express the search for such supermodels tends to significantly increase the problem size.

### *5.3. Artificial Immune Systems*

A very different approach to off-line/on-line algorithms for robust scheduling is preliminary work that examines the creation of an artificial immune system [21,22]. The approach rests on the conjecture that the conditions underwhich a new schedule is needed in response to some unexpected event in a factory are to some extent predictable. For example, there may patterns of customer orders and resource loads (perhaps depending on seasonality), there may be particular machines that breakdown, or there may be factories from which deliveries are typically late (or early). Furthermore, these patterns of events may have corresponding actions that can be taken during rescheduling to minimize their impact. The reactions are actually the partial schedules that are put in place

during rescheduling. Therefore, the authors' goal, is to use genetic algorithms (GAs) to evolve a set of partial schedules (based on historical schedules of a factory) that can be used as building blocks to respond to an unexpected event. These schedule pieces encode some specific domain dependent knowledge about reasonable schedules for the factory and therefore will significantly reduce the search space required for rescheduling.

A set of schedule fragments is therefore evolved, off-line, for each resource using a GA. The fragments are represented by a sequence of activities that can execute on that resource plus a "wildcard" activity. The fitness criteria for the evolution involves the matching of the fragment against historical schedules. A match occurs if some sub-sequence of activities in a fragment is found to exist in an historical schedule. The wildcard activity matches any activity allowing the possibility of evolving more complex, non-contiguous patterns. The on-line phase is then to combine the evolved sequences to form schedules that are reactions to the unexpected state of the factory. Since similar states are likely to have been encountered in the past, a combination of the evolved sequences is likely to encode a good schedule.

The experimental results from this technique have so far focused on the creation of the schedule fragments and the extent to which these fragments could be used to "recognize" schedules (both those that they had been evolved with and new factory schedules). It has been shown that, depending on parameter settings during the evolution, fragments could be successfully evolved to match portions of the existing schedules.

While the artificial immune system concept has a certain appeal, it is, at this stage, far too preliminary even to compare against other techniques. It would seem to still be a significant step from matching valid schedules to being able to react to such standard types of unexpected events as machine breakdowns and duration uncertainties.

### *5.4. Markov Tasks Sets*

As discussed above in Section 4.2, work by Meuleau et al. [33] examined the modeling of resource allocation problems with Markov Decision Processes (MDPs). In order to solve problems beyond the tractability limit of standard MDP techniques the authors also developed an off-line/on-line approach in which the problem is first decomposed so that the allocation for each target is solved to

optimality independently and then the individual solutions are greedily integrated to find a good, but not necessarily optimal, global solution.

In the first, off-line, phase, an MDP for each target is solved independently using standard MDP solving techniques (e.g., dynamic programming) and ignoring constraints on the overall number of weapons as well as on the number of planes per state. These solutions result in a set of optimal weapon allocations that are integrated in the second, on-line, phase. In this phase, the set of actions with respect to each target are calculated and executed. The result of execution in terms of the number of remaining weapons and the status of each target (i.e., destroyed or not) is then observed and used to find the set of actions in the next state.

Given a the set of undamaged, available targets in a state, the set of actions is found in a greedy manner by using the component solutions to define a marginal expected utility of assigning a weapon to a target (given the number of weapons already assigned). The weapons are assigned one-by-one to the target with highest marginal expected utility until that utility is less than or equal to zero or until the limit on the number of weapons is reached.

This schedule, however, may still fail to satisfy the limit on the number of planes that can be used in a single state. A greedy deallocation-reallocation technique is then used to satisfy the plane constraints. The authors calculate the expected marginal decrease in utility from deallocating each plane (individually) at the state in question and then reallocation its weapons to targets at some future state. This utility is found again using the component solutions and a single step lookahead of the algorithm originally used to allocate weapons. That is, for each plane, its weapons are deallocated from the previous state (incurring a decrease in utility) and then the greedy allocation algorithm is run to allocate these available weapons to some subsequent states (increasing utility). The plane whose deallocation results in the smallest net decrease in expected utility is deallocated and the reallocation already computed in the lookahead is put in place.

Empirical evaluation of this technique against a single global MDP as well as two variations on a greedy technique (based also on the component solutions) reveals that in problems that could be solved by the single MDP, both the MTS model and one of the greedy models resulted in competitive solutions in terms of quality (i.e., expected utility). On larger problems, the MTS model performs substantially better than the two greedy variations.

The authors point to three insights that allow the solving of large MDPs in the fashion demonstrated:

1. the ability to decompose the the MDP into pseudo-independent sub-processes

2. the use of the solutions to the sub-processes to guide in the construction of a global solution

3. the use of an online policy to remove the need to reason about future contingencies

### *5.5. Discussion*

Delayed and least commitment scheduling together with problem decomposition techniques such as Markov Task Sets would appear to be critical tools in dealing with uncertainty. Indeed, the form of decomposition used in the MTS work can be seen as a form of delayed commitment scheduling: the integration of the sub-problems is left until execution time when the information about the success or failure of each task is available. Unless care is taken, however, these approaches can suffer from the same drawback as the contingency-based techniques: the off-line schedule may not produce enough information, for other, dependent nodes in the supply chain to confidently create local schedules. This is most apparent in the MTS work as the off-line schedules for each target may have little relation to the integrated schedule. The goal in the off-line phase must be, as noted by Wu et al. [47], to make a sufficiently detailed schedule so as to serve as a basis for dependent entities in the supply chain while maintaining enough flexibility to deal with disturbances at execution time. Though work reviewed here has begin to address this goal, it remains a challenging problem.

The concept of supermodels is certainly attractive however its applicability to real scheduling problems seems limited. While scheduling problems can be encoded in SAT [8], the results noted concerning the existence of supermodels only for underconstrained problems would seem to discount many real-world scheduling problems. Nonetheless, the idea of off-line algorithms creating solutions (or even partial solutions) with guarantees in the amount of work necessary in the on-line phase is certainly valuable. Perhaps, the concept, if not the actual implementation of supermodels, can be applied to scheduling.

## 6. Rescheduling

Although there is much that can be done to ensure that a predictive schedule can absorb uncertainties, even the most robust schedules can be overwhelmed by an unstable, rapidly changing environment. In a dynamic environment, a variety of unexpected events are continually occurring and any schedule may in practice be subject to frequent revision to reflect this. Scheduling in such an environment is thus an ongoing and continuous process. Furthermore, even in a less dynamic environment, it is unlikely that a predictive schedule will be able to cope with all possible sources of uncertainty: some events will be judged too unlikely to plan for and so when they happen some sort of rescheduling will have to be done.

The approach to rescheduling considered in this section is to "repair" the previous predictive schedule to take into account the unexpected events that have occurred. Repairs may take the form of simple, fast control rules to make decisions within some real time execution constraints and to tend to minimize the perturbation to the original schedule.

Examples of constraint based systems which perform rescheduling are ISIS [14], MicroBoss [39,38], SONIA [29], OPIS [43,41,42], DSS [24] and Gerry [50,49]. Leon et al [30] propose a predictive and reactive control system which treats recovering from disruptions as a game playing problem against nature.

A number of techniques have been proposed to reschedule with minimal perturbation. ISIS [14] performs rescheduling with preferences on the new solution being the values of the variables in the old solution. Some researchers have suggested using iterative repair to perform rescheduling. Zweben et al. [49] claim that a disadvantage of constructive rescheduling techniques is that the task of determining the set of activities to reschedule is not straightforward. Iterative repair techniques need not remove any tasks from the schedule to perform rescheduling, so are not faced with this problem. In contrast, Sadeh et al. [39] claim that a drawback of iterative repair techniques is that in the process of resolving existing conflicts they may introduce new conflicts which in turn require more repairs: such iterative behavior can sometimes lead to myopic decisions which can be difficult to deal with. Although we are unaware of any theoretical or empirical work which compares these two approaches, some theoretical analysis for iterative repair does show that if a candidate, inconsistent solution is "sufficiently" close to a consistent solution in terms of hamming distance, there is a high probability that iterative repair will converge to the consistent solution

[34,28]. However there is no way of knowing *a priori* whether a solution is close enough for these results to apply.

In this section we consider three approaches to rescheduling:

1. MicroBoss [39] utilizes a range of control rules and uses them either in a reactive fashion to quickly restore consistency or as a procedure to identify which activities need to be fully rescheduled.
2. Sakkout et al. [12,13] formulate the rescheduling problem as a minimal perturbation problem, and perform full rescheduling using a combination of constraint programming and linear programming techniques with the goal of minimizing changes from the original schedule.
3. Wallace & Freuder [44] describe a CSP technique for generating stable solutions based on previous data of the causes of solution breakage. While this is not, strictly speaking, a rescheduling technique, it is a concept that may have applications for scheduling problems as well as the more general CSPs.

### *6.1. Reactive Scheduling in MicroBoss*

In the MicroBoss scheduler there are two levels of control within which disruptions to the schedule can be handled:

1. **Control level:** small disruptions which require fast responses are handled by simple control rules, such as "process the operation with the earliest scheduled start time first" or "when a machine is down, reroute critical jobs to any available equivalent machines".
2. **Scheduling level:** when more severe disruptions have occurred, the scheduling algorithms within MicroBoss are used to repair or reoptimize the schedule from a more global perspective, while still continuing to attend to more immediate decisions.

An important issue is deciding which disruptions should be handled by which level. There is a tradeoff, depending on the conditions within which a breakdown occurs, between the slower, better decisions of the scheduling level and the faster, more local decisions at the control level.

In repairing a schedule at the scheduling level, MicroBoss first selects a number of activities to unschedule, then reschedules them using the full MicroBoss micro-opportunistic search procedure. The rescheduling occurs in the context of the remaining current schedule which is not invalidated by the schedule breakage.

A number of conflict propagation procedures are proposed that can be used either at the control level to reschedule activities or at the scheduling level to identify the set of activities to be rescheduled. It might be the case the the identified activities are insufficient to enable a new solution to be generated. In such a case, this set of activities can be expanded incrementally until a solution is found. An example of the one of these propagation procedures, the *Right Shift Rule*, is illustrated in Figure 2.

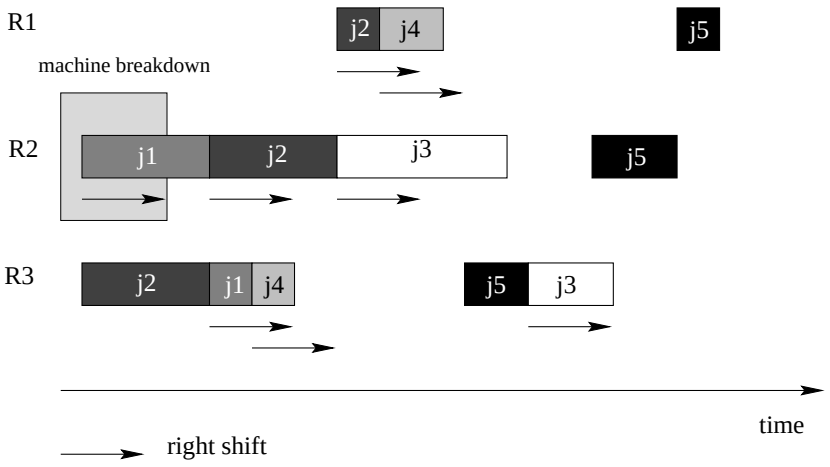


Figure 2. The Right Shift Rule

In this example there a three resources, $R_1$, $R_2$ and $R_3$, and five jobs $j_1$ to $j_5$ executing on these resources. Resource $R_2$ breaks down near the beginning of the schedule. The Right Shift Rule can be applied at the control level by moving forward in time all those activities which are affected by the breakdown, either because they were executing on the resource when the breakdown occurred or as a result of precedence constraints. This technique can produce poor solutions at the control level as it does not resequence activities. For instance, the activity in job $j_4$ on resource $R_3$ can still execute at its current time, even though in the original schedule it came after activity $j_1$ on this resource. Therefore, better schedules can be achieved by using the rule to identify the set of activities to be rescheduled by the full MicroBoss scheduler at the scheduling level, where resequencing can occur.

A number of control rules and procedures of varying complexity for identifying sets of activities to reschedule are presented in Sadeh et al. [39]. These are

evaluated on a set of randomly generated problems with or without bottleneck resources, with a single simulated machine breakdown. The experimental study found that the total rescheduling of all remaining activities produced the best quality solutions, but, unsurprisingly, resulted in the greatest disruption to the original schedule and took the longest time. However, it was shown that one of the more sophisticated activity identification procedures followed by rescheduling was able to find almost as good schedules as complete rescheduling of the remaining activities, while rescheduling 30% fewer activities.

### *6.2. Minimal Perturbation Rescheduling*

Often the goal in rescheduling is not just to restore consistency to the original schedule, but also to find a new schedule which deviates from the original schedule as little as possible. This is the problem of rescheduling with minimal perturbation. Minimal perturbation is important when the schedule serves as a basis for other planning activities in the supply chain (e.g., deliveries to and from other nodes). It is also important when we want to avoid "shop floor nervousness" caused by frequent changes to a schedule.

More formally, El Sakkout et al. [12,13] define the *minimal perturbation problem* as the need to reschedule a set of activities such that all resource and temporal constraints are respected and such that the sum of the absolute differences between the start time of each activity and the original start time of that activity (i.e., in the original schedule) is minimized.

Unimodular probing [12,13] is used to solve this problem by representing the temporal and resource constraints in a constraint programming solver, while representing the temporal constraints and the cost function as a linear program. In particular, the linear program does not represent the resource constraints. A branch-and-bound technique is used where, at each search node:

- The linear program is solved to generate a supra-optimal start time for each activity: minimizing the cost function subject to the temporal constraints.
- The constraint programming solver then analyses the optimal start times to evaluate the breakage, if any, of the resource constraints.
- One of the broken resource constraints is heuristically selected and a new precedence constraint is added between a pair of activities contributing to the resource breakage. The new precedence constraint is the branching step and it is added to both the constraint solver and the linear solver.

- Constraint propagation is used to derive new constraints implied by the new precedence constraint.

The unimodular probing technique was evaluated on a number of random benchmark problems and compared with a number of constraint solving techniques and mixed-integer programming (MIP). Given a fixed timeout, the unimodular probing technique was capable of proving optimality for many of the benchmark problems, where the constraint programming techniques failed to do so. MIP performed relatively well but was still inferior to unimodular probing.

### *6.3. Stable Solutions*

When rescheduling occurs frequently in an environment, it may be possible to gather information characterizing the frequency and impact of stochastic events as they occur. This information can be used later when rescheduling to move to more robust schedules. This idea has been investigated in the context of solving the recurrent dynamic CSP [44].

A dynamic CSP is one in which constraints may be added or removed over time and where the goal is to maintain a solution to the current model. In a recurrent dynamic CSP (RDCSP) [44], the changes tend to be temporary and there are differences in the likelihood of changes. For example, a value may be externally removed from a domain of a variable but later re-inserted and the likelihood that a particular value is removed is different from the likelihood that some other value is removed. Such assumptions can be understood from the perspective of scheduling by allowing the modeling of the fact that a resource may breakdown, removing it as a possible value for the resource requirement of an activity. However, after some time for repair, the resource will be reinserted as a possible value.

During the process of repeatedly solving the RDCSP over time, the authors gather statistics for the frequency at which variables and values of the CSP which become unavailable. These statistics are used in iterative repair technique. The technique, based on Min-Conflicts hill-climbing [34], repairs the assignments of variables which violate problem constraints. The value to select for a variable when making a repair is guided by an evaluation function which rates highly those values which violate the fewest constraints while also taking into account how often in the past each value has become unavailable. During the search for a solution, good values for a variable which become unavailable often might

not be preferred to other values which violate more constraints but which are more reliable. A number of issues are investigated with respect to creating an evaluation function which takes into account both constraint violations and value reliability.

### *6.4. Discussion*

It seems clear that, regardless of the techniques to generate a robust, predictive schedule, if any, some rescheduling will need to be done at execution time. The, perhaps unachievable, goal of such rescheduling is to quickly deliver a new schedule that is optimal both in terms of the relevant measures of schedule quality and in terms of minimal deviation from the original schedule.

This rescheduling may be as simple as the Right Shift Rule or may involve a complete rescheduling of all activities that have not yet executed. As the work using MicroBoss indicates, the appropriate trade-off is far from clear and, indeed, depends on the external situation at the time of rescheduling. In this context, the unimodular probing work provides provably minimal deviation rescheduling, however, does not address the requirement of fast rescheduling. One possibility that may preserve many of the advantages of this approach is to reschedule near-term activities using a simple rule and then, after execution of the schedule has recommenced, to reschedule the later activities with unimodular probing or some other technique that can guarantee optimality.

Interestingly, the stable solution concept shares some of the approach of the artificial immune system work discussed above: both attempt to gather statistics (implicitly or explicitly) about the schedules and events that have occurred in the factory in order to guide the search for a reaction to subsequent events. However, it is unclear how well stable solutions fit into the scheduling framework. For example, if a value in the domain of a variable is used to represent a possible resource assignment for an activity and this resource frequently breaks down, a stable solution will tend to avoid assigning activities to the resource. Given a resource load that requires the resource to execute some activities, it is not clear that this implicit reasoning about uncertainty will result in robust solutions. It may be better to explicitly reason about the uncertainty associated with the resource and so, for example, assign it to activities that are otherwise uncritical: low priority activities that are not on the critical path and that are part of jobs which have low penalties for missing due dates.

## 7. Discussion

Schedules are executed in uncertain environments. At its most general, the ability to deal with uncertainty in scheduling is the ability to achieve high quality schedule execution despite the occurrence of unforeseen events. The quality of schedule execution has a variety of standard measures in industry and research: minimal makespan, maximal throughput, minimal idle time, maximum on-time delivery rate, etc. While this definition of dealing with uncertainty does not necessarily preclude dealing with uncertainty solely at schedule execution time, the theoretical difficulty of scheduling, the need for quick decision-making, and the dependencies among schedules at different nodes in a supply chain necessitate the incorporation of robustness into predictive schedules.

Most of the work we have reviewed in this paper deals with schedule robustness. While there is a general, informal understanding of what robustness is (i.e., the ability of a predictive schedule to cope with unforeseen events), there is no agreement on a formal definition of schedule robustness. Based on the work reviewed here, however, we believe that the reason for such a lack of agreement is that there is no formal definition that will fit the myriad of ways in which robustness can be defined in specific systems. A formal definition that limits robustness to a particular form will only be applicable in to applications that precisely meet that definition. Any system that hopes to address robustness in scheduling, in general, will have to allow a different, specific definition of robustness in different situations to which it is applied.

While robust predictive schedules are necessary, they do not remove the requirement for execution time reasoning. Some reasoning, even if it is as simple as the Right Shift Rule, will have to be done at execution time. This reasoning is able to take advantage of truly up-to-date information, however it is highly constrained by the amount of time it can spend in reasoning. Furthermore, regardless of the ability to generate robust predictive schedules, there will be unexpected events that are too expensive and too unlikely to account for. When such events occur, some execution time rescheduling is necessary.

In summary, then, a system that is able to deal with uncertainty in scheduling is one for which:

- the specific manifestation of uncertainty and definition of robustness can be specified.
- uncertainty can be taken into account in generation of predictive schedules.

- the occurrence of unexpected events can be reacted to at execution time.

All of these issues are currently still open questions. Obviously, work that contributes to the an understanding of these questions, even in isolation, is valuable, however we view the above description as the overall goal for such research.

### *7.1. Techniques for Dealing with Uncertainty in Scheduling*

This paper has surveyed research from a number of different areas that all bear on the issue of dealing with uncertainty in scheduling. In this section, we comment specifically on approaches which appear to have the most and least promise. These comments are, of course, to some extent subjective, however, we believe they are useful at the very least as an indication of where future research directions may lie.

- Probabilistic reasoning appears to be an important step in terms of bringing a level of formality to reasoning about uncertainty in scheduling. Even if it is shown that exact probabilistic guarantees of schedulability are intractable in practice, the very existence of a formal theory will provide significant support on which approximation techniques can be based. Of the approaches reviewed in this paper and of which we are aware, it appears to be the only one which may be able to be developed into a formal theory.
- As shown in the fault tolerant real-time scheduling field, redundancy techniques can provide the tools for a significant simplification of both predictive robustness measures and the complexity of rescheduling. It may be possible to achieve some degree of formality and still allow tractable schedule generation by combining redundancy techniques with probabilistic reasoning.
- Contingent scheduling techniques do not appear promising both from the perspective of the combinatorics of dealing with contingencies from multiple resource and from the need to have some "firm" information from the predictive schedule from which other dependent nodes in the supply chain can form their own schedules.
- An off-line/on-line approach to dealing with uncertainty is explicit in the comments in the our delineation of a system which addresses uncertainty in the previous section. There is clearly a need for both off-line, predictive techniques and on-line, reactive techniques of dealing with uncertainty. The challenge is to have an off-line technique which provides enough information for other nodes

in the supply chain without becoming inflexible while having an on-line phase that can quickly take near-optimal decisions.

- The technique of decomposing scheduling problems such as seen in the Markov Task Sets and in delayed/least commitment scheduling would seem to be a powerful tool to deal with the complexity of large scheduling problems. There is still a danger that unless some of the reintegration of sub-problems is done off-line, the information produced by the solving of the sub-problems will not be sufficient for other nodes in the supply chain. Furthermore, it should be noted that delayed commitment scheduling may, in fact, make it more difficult to perform probabilistic reasoning about a predictive schedule. Not only must the probabilistic reasoning cover the various external probabilities, it must also reason about all the different schedules that can still be produced at execution time.
- The minimal perturbation rescheduling problem forms a standard concerning what can be achieved at execution time by an on-line algorithm. While it is unlikely that such reasoning is possible in the limited time available, it may be that similar problem solving can be incorporated at nonetheless. As suggested above, perhaps some subset of rescheduling decisions can be made with a quick, sub-optimal rule so that the schedule can continue to execute while the other activities disturbed by a disruption are optimally rescheduled.

### *7.2. Open Issues*

Above we implicitly presented a number of open issues in terms of how uncertainty can be dealt with in scheduling. In this section, we explicitly present a number of other open issues that can be distilled from the work that has been surveyed here.

- Most of the research into uncertainty in scheduling addresses only one source of uncertainty: usually either machine breakdowns or uncertain durations. As noted at the beginning of this paper, the sources of uncertainty in real world scheduling are significantly more varied and, indeed, it would be unusual that a factory would experience only one form of uncertainty. A challenge, then, is not only to investigate other sources of uncertainty and how they impact the reasoning techniques discussed here, but also how the combination of sources of uncertainty (e.g., machine breakdown plus durational uncertainty plus modified due dates) can be dealt with.

- There is no understanding of the tradeoff between robustness and other scheduling performance metrics, such as tardiness and work in progress. While it seems clear that, at least with redundancy techniques, robustness will tend to increase tardiness, in a factory that runs robust schedules, the opposite may happen: because the management has a more accurate picture of when orders may be realistically delivered, they will be more accurate in the due dates that they promise to customers.
- There is no work that we are aware of that extends the notion of robustness to the problem of rescheduling. For example, minimal perturbation rescheduling focuses on the start and end times of the activities in the previous schedule. If such a technique or, indeed, other repair techniques, pay no attention to the robustness of the solution which they generate, an increasingly brittle schedule may result. Such a situation may result in catastrophic schedule failure in response to a relatively minor unexpected event. Clearly in an evolving system where a schedule is going to be repaired multiple times, the robustness of the repaired schedule, so that future repairs can be made, must be taken into account.[4]

## 8. Conclusion

After conducting a field study of a number of real world job shops, McKay et al. [32] are scathing about the state of job shop scheduling research: "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". Although they do not claim that no job-shop would benefit from the scheduling research conducted over the last few decades, a particular industry would need to have a number of deterministic characteristics for such research to be useful. McKay et al. believe that a new model of scheduling is needed, which takes into account conditions on the shop floor/operating environment and models how people use the schedule in practice.

The survey by McKay et al. was carried out ten years ago. Scheduling research has advanced considerably since then, especially within the area of constraint directed scheduling. Some of this research has made its way into the real world, where a number of successful commercial scheduling systems are now be-

[4] Thanks to Amedeo Cesta and Angela Oddi for this comment.

ing widely deployed (e.g. ILOG Scheduler, Cosytec CHIP, i2 Rhythm). Most of the research dealing with uncertainty reported in this survey has been carried out within the last few years. We have discussed research which deals with known and unknown uncertainties, as well as systems which learn about uncertainties in the scheduling environment. However, in general, we believe that the findings of McKay et al. still hold: there appears to be very little *understanding* or *characterization* of dynamic, uncertain scheduling environments. Despite this, the potential gains from improving our understanding can have a significant impact in the real world. The research surveyed in this article is a first step in that direction.

## 9. Acknowledgements

Much of this survey was written while the first author was working in the Enterprise Integration Laboratory at the University of Toronto. This survey was partially carried out as a part of a project to investigate uncertainty in constraint-directed scheduling funded by the Materials and Manufacturing Council of Ontario. This survey benefitted from comments and discussions with Hani El Sakkout, Amedeo Cesta, and Angelo Oddi.

## References

[1] J. C. Beck, A. J. Davenport, E. D. Davis, and M. S. Fox. The ODO project: Toward a unified basis for constraint-directed scheduling. *Journal of Scheduling*, 1(2):89–125, 1998.

[2] 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.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[4] 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.

[5] A. Cesta, A. Oddi, and S. F. Smith. Profile-based algorithms to solve multiple capacitated metric scheduling problems. In R. Simmons, M. Veloso, and S. F. Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 214–223, Menlo Park, CA, 1998. AAAI Press.

[6] C. C. Cheng and S. F. Smith. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research, Special Volume on Scheduling: Theory and Practice*, 70:327–378, 1997.

[7] W. Y. Chiang and M. S. Fox. Protection against uncertainty in a deterministic schedule. In *Proceedings of the International Conference on Expert Systems for Production and Operations Management*, 1990.

[8] J. M. Crawford and A. B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1092–1097, 1994.

[9] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in satisfiability problems. *Artificial Intelligence*, 81:31–57, 1996.

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

[11] M. Drummond, J. Bresina, and K. Swanson. Just-in-case scheduling. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1098–1104, Menlo Park, CA, 1994. AAAI Press/MIT Press.

[12] H. El Sakkout, T. Richards, and M. Wallace. Unimodular probing for minimal perturbance in dynamic resource feasibility problems. In *Proceedings of the CP-97 Workshop on Dynamic Constraint Satisfaction*, 1997.

[13] H. El Sakkout, T. Richards, and M. Wallace. Minimal perturbance in dynamic scheduling. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98)*, 1998.

[14] M. S. Fox. *Constraint-directed search: a case study of job-shop scheduling.* Morgan-Kaufmann Publishers Inc., 1987.

[15] 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.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[17] S. Ghosh. *Guaranteeing fault tolerance through scheduling in real-time systems.* PhD thesis, University of Pittsburgh, 1996.

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

[19] M.L. Ginsberg, A.J. Parkes, and A. Roy. Supermodels and robustness. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

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

[21] E. Hart and P. Ross. The evolution and analysis of a portential antibody library ror job-shop scheduling. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 185–202. McGraw-Hill, 1999.

[22] E. Hart and P. Ross. An immune system approach to scheduling in changing environments. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and

R.E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 1559–1565. Morgan Kaufman, 1999.

[23] W. S. Herroelen and E. L. Demeulemeester. Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems. In P. Chretienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors, *Scheduling theory and it applications*, chapter 12. John Wiley & Sons, Ltd., 1995.

[24] 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.

[25] D. Kjenstad. *Coordinated supply chain scheduling.* PhD thesis, Deparment of Production and Quality Engineering, Norwegian University of Science and Technology, Trondheim, Norway, 1999.

[26] A. Kott and V. Saks. Continuity-guided regeneration: An approach to reactive replanning and rescheduling. In *Proceedings of FLAIRS-96*, 1996.

[27] A. Kott and V. Saks. A multi-decompositional approach to integration of planning and scheduling – an applied perspective. In *Proceedings of the Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, Pittsburgh, USA, June 1998.

[28] H.C. Lau. Probabilistic analysis of local search on random instances of constraint satisfaction. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*, pages 195–199, 1996.

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

[30] V.J. Leon, S.D. Wu, and R.H. Storer. A game-theoretic control approach for job shops in the presence of disruptions. *International Journal of Production Research*, 32(6):1451–1476, 1994.

[31] V.J. Leon, S.D. Wu, and R.H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.

[32] K.N. McKay, F.R. Safayeni, and J.A. Buzacott. Job-shop scheduling theory: What is relevant? *Interfaces*, 18(4):84–90, 1998.

[33] 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.

[34] S. Minton, M. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[35] D. Neiman, D. Hildum, V. Lesser, and T. Sandholm. Exploiting meta-level information in a distributed scheduling system. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 394–400,, Seattle, WA, 1994.

[36] W. P. M. Nuijten and E. H. L. Aarts. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, 1997.

To appear.

[37] M.J. Ringer. Time-phased abstractions for combining predictive and reactive scheduling methods. In *Workshop Notes of the AAAI-92 SIGMAN Workshop on Knowledge-Based Production Planning, Scheduling, and Control*, pages 121–126, 1992.

[38] N. Sadeh. *Lookahead techniques for micro-opportunistic job-shop scheduling.* PhD thesis, Carnegie-Mellon University, 1991. CMU-CS-91-102.

[39] N. Sadeh, S. Otsuka, and R. Schelback. Predictive and reactive scheduling with the Micro-Boss production scheduling and control system. In *Proceedings of the IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling, and Control*, pages 293–306, 1993.

[40] S. F. Smith and M. Becker. An ontology for constructing scheduling systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering.* AAAI Press, 1997.

[41] S.F. Smith. OPIS: A methodology and architecture for reactive scheduling. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, chapter 2, pages 29–66. Morgan Kaufmann Publishers, San Francisco, 1994.

[42] S.F. Smith. Reactive scheduling systems. In D.E. Brown and W.T. Scherer, editors, *Intelligent scheduling systems.* Kluwer Press, 1995. Available on line from http://www.cs.cmu.edu/afs/cs/project/ozone/www/icll-pubs.html.

[43] S.F. Smith, N. Keng, and K.G. Kempf. Exploiting local flexibility during execution of precomputed schedules. In A. Famili, D.S. Nau, and S.H. Kim, editors, *Artificial Intelligence Applications in Manufacturing*, chapter 12, pages 277–292. AAAI Press/MIT Press, 1992.

[44] R.J. Wallace and E.C. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP-98)*, pages 447–461, 1998.

[45] T. F. Wallace and J. R. Dougherty, editors. *APICS dictionary.* APICS, sixth edition, 1987. ISBN 0-935406-90-S.

[46] W.P. Wellman. Challenges of decision-theoretic planning. In *Proceedings of the AAAI Spring Symposium on Foundations of Automatic Planning*, 1993.

[47] S.D. Wu, E. Byeon, and R.H. Storer. A graph-theoretic decompostion of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, 1999.

[48] P.R. Wurman and M.P. Wellman. Optimal factory scheduling using stochastic dominance A*. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 1996.

[49] M Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, chapter 8, pages 241–256. Morgan Kaufmann Publishers, San Francisco, 1994.

[50] M. Zweben, M. Deale, and R. Gargan. Anytime rescheduling. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 251–259. Morgan Kaufmann, SamMateo, California, 1990.