J Sched (2009) 12: 315–344
DOI 10.1007/s10951-008-0080-x

A theoretic and practical framework for scheduling in a stochastic environment

Julien Bidot · Thierry Vidal · Philippe Laborie · J. Christopher Beck

Received: 17 May 2007 / Accepted: 4 July 2008 / Published online: 23 August 2008

Abstract There are many systems and techniques that address stochastic planning and scheduling problems, based on distinct and sometimes opposite approaches, especially in terms of how generation and execution of the plan, or the schedule, are combined, and if and when knowledge about the uncertainties is taken into account. In many real-life problems, it appears that many of these approaches are needed and should be combined, which to our knowledge has never been done. In this paper, we propose a typology that distinguishes between proactive, progressive, and revision approaches. Then, focusing on scheduling and schedule execution, a theoretic model integrating those three approaches is defined. This model serves as a general template to implement a system that will fit specific application needs: we introduce and discuss our experimental prototypes which validate our model in part, and suggest how this framework could be extended to more general planning systems.

Keywords Scheduling · Planning · Uncertainty · Robustness · Combinatorial optimization · Constraint programming · Simulation · Flexibility · Stability

This article is a longer and extended version of a conference paper which appears at IJCAI'07 (Bidot et al. 2007).
J. Bidot is partially supported by *Convention Industrielle de Formation par la REcherche* 274/2001.

J. Bidot (✉)
Universität Ulm, Ulm, Germany
e-mail: julien.bidot@uni-ulm.de

T. Vidal
IRISA-INRIA, Rennes, France
e-mail: thierry.vidal@irisa.fr

P. Laborie
ILOG S.A., Gentilly, France
e-mail: plaborie@ilog.fr

J.C. Beck
University of Toronto, Toronto, Canada
e-mail: jcb@mie.utoronto.ca

1 Introduction

Scheduling usually starts from a given set of activities, which must satisfy a set of temporal and resource constraints, and search for a schedule in which precise start times are set and precise resources are allocated to each activity. Very often an optimal schedule is actually looked for, taking into account some optimization criteria (minimal makespan, minimal resource consumption, etc.). In Job-Shop Scheduling, usually addressed by Operations Research techniques, the set of activities to schedule comes from the so-called jobs, consisting of sequences of predetermined activities, which must be included in a shop, accounting for resource demands. In Artificial Intelligence planning, the set of activities is produced by the planning search engine, which reasons about goals to achieve, available generic activities, and causality relations among them, to provide a plan of activities (i.e., instantiated activities ordered through their preconditions and effects), which usually disregards resource usage conflicts and may not commit to a precise complete ordering.

But the frontier between Artificial Intelligence planning and scheduling is not that clear, especially when time, resources, and uncertainty must be considered (Smith et al. 2000). This is why people are more and more interested in a more general planning and scheduling framework in which all kinds of planning and scheduling decisions to make to reach the eventually executed schedule are considered all at once: we propose in this paper to use the general term of *schedule generation* to refer to such an extended framework.

It is particularly true when uncertainties come into the picture: in that case, in both Job-Shop Scheduling and Artificial Intelligence planning, even the usual strict distinction between the generation and the execution of the schedule must be relaxed. Classical approaches for solving planning and scheduling problems are both predictive and deliberative, in that a schedule is designed offline and then sent to the execution controller, which can execute it online in a straightforward manner, but with no ability to reconsider anything. However, in practical applications, we have to plan or schedule with incomplete, imprecise, and/or uncertain data: simply executing a strictly and completely determined predictive schedule is not sufficient, as there is a high chance that such a schedule will not fit the real situation that will arise. As we are going to see it, in stochastic environments, both planning (i.e., which activities to choose) and scheduling (i.e., when to start them and on which resources) decisions might either be anticipated at generation time, or postponed until execution time, when some uncertainties are resolved.

That is exactly the focus of this paper: we are interested in the way generation and execution of the effective schedule should be jointly considered to best fit uncertainty management needs. This means we are not deeply concerned with the actual planning search or job-shop scheduling techniques that are used to generate that schedule. The questions we want to address are: How a global generation/execution loop should be designed? If available, should knowledge about possible deviations be used, and if yes, how could one integrate them in the solution process? How much one should commit to the predictive decisions made offline? Should online revision of such decisions be allowed? To what extent? Alternatively, how much of the decision process may be postponed and only be taken online? How do our choices influence the online efficiency, memory consumption, or the quality of the eventually executed schedule? In other words, our goal is to study planning and scheduling under uncertainty in terms of how and when decisions are made.

In this paper, we have chosen to focus on a basic optimal scheduling problem: we want to allocate resources and assign start times to a set of activities, so that temporal and resource constraints are satisfied and so as to optimize a given quality metric. We will also use constraint-based techniques to validate our model. However, the work presented here can be extended to a more general schedule generation framework in which planning decisions are integrated, since we are interested in when and how various decisions get made, and not what those decisions are. Somehow, as we will see, some limited planning concerns already fit in the picture, since we allow alternative sequences of activities to be generated, the selection being done at execution time, in a conditional planning like manner.

After having proposed some basic definitions in Sect. 3, we provide a brief review of some relevant work in planning and scheduling to exhibit a thorough classification of the techniques for planning and scheduling under uncertainty, discussing their strengths and weaknesses. We show that in real-life applications, mixing those techniques within a single system appears to be highly desirable. For that purpose we propose (in Sect. 4) a new conceptual model encompassing both the generation and the execution of schedules and in which a variety of techniques for dealing with uncertainty can be concurrently implemented. Section 5 presents experimental prototypes that validate our model in part. Finally, we discuss how our framework can be extended to planning in Sect. 6.

2 Some basic definitions

Before presenting our formal model in Sect. 4, we informally describe the problem we are interested in, only to introduce basic concepts and properties that are relevant when addressing schedule generation and schedules in an uncertain environment.

A standard scheduling problem comprises a set of activities and a set of resources. Each activity has a duration, there are temporal relations between activities, and each resource has a limited capacity. The objective is to assign resources and times to activities given temporal and resource constraints. To achieve this, we have to make scheduling decisions which are typically: choosing allocations, choosing sequences, and setting activity start times. We endow the traditional scheduling system with the capability of representing alternative subsets of partially-ordered activities, among which only one must be chosen and executed.

In general, scheduling problems are also optimization problems: typical optimization criteria are makespan, number of tardy activities, tardiness or allocation cost, etc.

If we assume an execution environment without uncertainty, one usually generates a schedule offline that is then executed online without any problem. There are, however, many possible sources of uncertainty in scheduling, e.g., some activity durations or some resource capacities are imprecise (as resources may break down).

We now give some definitions to avoid ambiguity of terms commonly used by different communities.

Definition 2.1 (Complete schedule) A complete schedule is the solution of a scheduling problem, i.e., all decisions are made: activity start times are set, resource allocations are done, sequencing decisions are made, and no alternative subsets of partially ordered activities remain.

Definition 2.2 (Flexible schedule) A flexible schedule is an incomplete schedule: decisions have still to be made. Constraints are associated with a flexible schedule to restrict the set of complete schedules that can be derived from it.

Definition 2.3 (Conditional schedule) A conditional schedule is a schedule in which distinct alternative subsets of partially ordered activities can be modeled. The schedule contains a condition to test for each choice between such alternatives at execution time.

Definition 2.4 (Predictive schedule) A predictive schedule is a schedule that is generated before the end of its execution. A predictive schedule is generated before or during execution.

We may have to modify predictive schedules to adapt them to online situations we do not know beforehand.

Definition 2.5 (Executable schedule) A schedule is executable at time t if and only if it does not violate any constraint known at time t.

Definition 2.6 (Adaptive scheduling system) An adaptive scheduling system is a system that is able to generate a new executable schedule whenever the currently executing schedule is no longer executable.

Definition 2.7 (Robust predictive schedule) A predictive schedule is said to be robust if the quality of the eventually executed schedule is close to the quality of the predictive schedule. More formally, we can state that a predictive schedule with quality q_{pred} is ϵ-robust for a given ϵ and optimization criterion if the quality, q_{exec}, of the eventually executed schedule is such that $(1-\epsilon) \times q_{\text{pred}} \le q_{\text{exec}} \le (1+\epsilon) \times q_{\text{pred}}$ given an execution controller and known online perturbations.

The possible values of ϵ are in $[0, 1]$. We use ϵ values to compare the robustness of predictive schedules: the smaller the ϵ value, the more robust the predictive schedule.

Definition 2.8 (Stable predictive schedule) A predictive schedule is said to be stable if the decisions made in the eventually executed schedule are close to the decisions made in the predictive schedule. More formally, we can state that a predictive schedule containing the set of decisions DES_{pred} is Δ-stable for a given Δ if the set of decisions, DES_{exec}, in the eventually executed schedule is such that $(1-\Delta) \times |\text{DES}_{\text{pred}}| \le |\text{DES}_{\text{pred}} \cap \text{DES}_{\text{exec}}|$ given an execution controller and known online perturbations.

The possible values of Δ are in $[0, 1]$. We use Δ values to compare the stability of predictive schedules: the smaller the Δ value, the more stable the predictive schedule.

Stability and robustness may be independent as long as the quality of the predictive schedule is low when compared to the optimal one: it will not be very difficult to maintain such a low quality online, still changing no decision. But as soon as the schedule quality is high enough the two may quickly become antagonistic. A flexible schedule can be stable but not robust, which means that we do not change decisions but its actual quality (obtained after execution) deviates from its predictive quality (i.e., the quality we estimate before execution). On the contrary, a flexible schedule can be robust but not stable in the case where we change decisions online to keep its effective quality after execution close enough to its predictive quality (i.e., the quality that was estimated before changing decisions).

3 Classification

In this section, we concisely describe a taxonomy of techniques for scheduling and planning under uncertainty that is independent of any specific representation or reasoning technique. Such classifications have already been done, especially in the Operations Research community (as in Billaut et al. 2007; Herroelen and Leus 2005), but none is totally satisfactory to our needs, since they only distinguish between offline and online techniques: proactive techniques take into account knowledge about uncertainty to make decisions offline, while reactive techniques are used online to adapt the current schedule when an unpredicted event occurs such as a machine that breaks down. We go beyond this distinction and consider issues such as how, when, and what decisions are made, optimality requirements, etc.

3.1 Our taxonomy in brief

We will in the following distinguish between three main families of techniques, which refer to three main ways of balancing schedule generation and execution:

Proactive techniques Such techniques stick to the classical idea of building a global solution at generation time, which will never be reconsidered at execution time. It is proactive (and not only predictive) in that knowledge about the uncertainties is taken into account in order to generate more reliable schedules. To reach such a goal, one may

- Generate one complete *generic* schedule which is proved to cover most cases, i.e., to execute correctly in most of the possible situations that will arise at execution time.
- Generate a *flexible* solution, i.e., in which some decisions have not been made and are postponed until execution time.
- Generate a *conditional* solution, i.e., a solution in which various mutually exclusive decisions are developed, the one being effectively chosen being dependent on some condition which will only be observed at execution time.

Revision techniques Such techniques generate a complete schedule in the classical predictive way, and at execution time, whenever that solution does not fit the observed situation, it is revised, i.e., some of the decisions already made are modified through online schedule regeneration.

Progressive techniques Also known as continuous, such techniques generate and execute in the classical way, but only locally, in the short term: a new part of the global schedule is generated online, either at predefined timestamps, or whenever a condition expressing that some uncertainties are resolved is satisfied.

3.2 Proactive techniques

A proactive technique takes into account the knowledge about uncertainty to produce schedules that are more robust, more stable, or both more stable and more robust than they would be without using this piece of information.

A first naive method for making a schedule insensitive to online perturbations is to produce offline a complete, predictive, robust schedule by taking into account the worst-case scenario. A more balanced view is to account for the knowledge on the possible deviations, and hence on all possible scenarios that may emerge online, to compute a complete but somehow *generic* solution that will be able to fit most of such situations. In other words, one generates a cautious complete schedule because one wishes to maximize the executability of such a solution online. That can be done, for instance, when one knows distributions (e.g., of probability) of the possible deviations, and can compute the solution that has the highest probability of success.

Different uncertainty models (probability distributions, possibility theory, etc.) can be used in proactive techniques for the representation of the problem, and for solving it (e.g., find the schedule that will have the highest probability that the makespan will not exceed a given value).

The research work of Dubois et al. (1993) is a good example of such *generic schedule* generation: they use fuzzy logic, tackling scheduling problems with fuzzy activity durations, and try to find a schedule that minimizes the possibility of performance less than a threshold. Standard search techniques are used but the constraint-satisfaction requirement is replaced by "reasonably sure that no constraint will be violated." This is a realistic approach that lies between accepting only schedules that are sure to work and accepting a schedule without taking into account possible deviations.

Another approach consists in introducing some flexibility in the schedule: only a subset of decisions are made offline with a search, and the rest are made online with no or only very limited search; this is a kind of least-commitment approach with respect to decision-making since we only make decisions when information is more precise, and/or more certain. Morris et al. (2001), for instance, maintain a simple temporal network representing a schedule with uncertain activity durations in which start times are not set: they provide algorithms to guarantee the executability of such schedules whatever the actual durations will be. Here we have a *flexible schedule*.

In the same vein, redundancy-based scheduling can also be considered as a proactive technique for scheduling. For example, Davenport, Gefflot, and Beck proposed an approach in which they add slack times to critical activities, i.e., the activities that are allocated on possibly breakable resources (Davenport et al. 2001). This work extended the Master's thesis of Gao (1995). The new temporally protected problem can then be tackled with techniques usually used to solve deterministic scheduling problems, e.g., constraint propagation algorithms can be used to make tree search more efficient. More precisely, activity durations are set to be longer than the original ones to generate a predictive schedule that can face possible machine breakdowns. The activity durations used are based on breakdown statistics, e.g., the mean time between failures or mean time to repair may be used.

Experiments on job-shop problems show that this technique significantly reduces the gap between the predictive quality and the effective quality, but it results in an increase of tardiness. However, the temporal protection given to activity a_1, which should allow the following activity a_2 allocated to the same resource to start earlier if no breakdown occurs, can be "lost" if a constraint prevents a_2 from starting earlier. This observation is at the outset of two methods, time-window slack and focused time-windows slack, presented by Davenport, Gefflot, and Beck who proposed to post additional constraints such that each activity has a minimum temporal slack. Simulation results show that these two methods are able to generate schedules whose tardiness is smaller than the tardiness of the schedules determined by the temporal protection technique. It is also shown that effective quality is predicted more accurately, i.e., predictive quality a priori (before execution) is closer to the actual quality, observed a posteriori (after execution). Although this approach is not based on real theoretical foundations, it is simple and pragmatic such that it can be readily applied to a real scheduling problem.

Generating partial-order schedules (POSs) is another example of such flexible approaches: a subset of sequencing decisions are made offline, and the remainder being made online with using a dispatching rule. For example, Policella et al. considered the problem of generating POSs for problems with discrete resources (Policella et al. 2004). They tackle resource-constrained project scheduling problems using filtering algorithms and propose two orthogonal procedures for constructing a POS. The first, which is called the resource-envelope-based approach, uses computed bounds on cumulative resource usage to identify potential resource conflicts, and progressively reduces the total set of temporally feasible solutions into a smaller set of resource-feasible solutions by resolving detected conflicts.[1] The second, referred to as the earliest-start-time approach, instead uses conflict analysis of an earliest-start-time solution to generate an initial fixed-time schedule, and then expands this solution to a set of resource-feasible solutions in a post-processing step. As might be expected, the second approach, by virtue of its more focused analysis, is found to be a more efficient POS generator based on experimental results.

Wu, Byeon, and Storer proposed another way of producing partial-order schedules (Wu et al. 1999). They identify a critical subset of decisions that, to a large extent, dictate global schedule performance. These critical decisions are made offline, and the rest of the decisions are done online. They address job-shop problems in which a weight and a due date are associated to each job. Before execution, they solve an Order Assignment Problem (OAP) optimally, i.e., they partition activities into groups such that original precedence constraints are respected, and introduce a set of precedence constraints to the problem between groups. Each possible assignment of activities to these groups is associated with a cost representing the minimum possible weighted tardiness given the precedence constraints imposed between the groups; the objective is to choose the assignment that minimizes this cost. After solving the OAP, we get a job-shop scheduling problem that can be solved to evaluate the partition (the solution of the OAP).

Another way to address flexibility is to build a *conditional schedule*: everything is set but with alternative branches. Just-In-Case scheduling illustrates this method perfectly. It is plainly a proactive technique for scheduling. Drummond, Bresina, and Swanson proposed a contingent scheduling method, called Just-In-Case (JIC) scheduling, applied to a real-world telescope observation scheduling problem where observation durations are uncertain (Drummond et al. 1994). The solution consists in building a contingent schedule that takes into account likely failures of the automatic telescope and that does no waste valuable observing time. The objective is to increase the percentage of the schedule that can be executed without breakage by assuming there is a scheduling algorithm for solving the deterministic problem, and probability distributions of the observation durations are known, i.e., the mean values and the standard deviations are known. The JIC scheduling proceeds as follows. Before execution, a schedule is generated assuming deterministic durations, then the algorithm identifies the activity with the highest break probability, the break point in the schedule is split into two hypothetical cases accordingly: one in which the schedule breaks and one in which it does not, and then the procedure finds an alternative schedule assuming the breakage. There are, of course, several likely breakages, so the procedure is applied several times. During execution, if there is no breakage, we keep executing the same schedule, otherwise, if the situation is covered by the contingent schedule, we switch to a new schedule. This is a successful, real-world solution. The online portion is trivial since we only have to switch to an alternative schedule. However, this method is applied to a one-machine scheduling problem, and the combinatorial complexity seems to hinder its generalization to multiple machine problems.

A last example of conditional models can be found in the work in planning made by Tsamardinos et al. (2003). They propose a conditional temporal model in which a plan contains alternative branches depending on the state of the world. They analyze the conditions under which observations of such states must be made prior to branching, taking into account quantitative temporal constraints (durations).

In addition, all the literature on Markov Decision Processes (Puterman 1994; Geffner 1998) (which we will not develop here) can be viewed as examples of such conditional proactive models for planning.

3.3 Revision techniques

Revision techniques consist in changing decisions during execution when it is necessary or desirable, e.g., we change decisions when the current predictive schedule becomes inconsistent, when estimated quality deviates too much from the predictive one, or, in a more opportunistic way, when a positive event occurs (for example, an activity finishes earlier than expected). In other words, we need an execution-monitoring system able to react and indicate when it is relevant to change decisions of the current predictive schedule. Note that such a procedure is called rescheduling, and a revision method is usually called predictive-reactive in the literature (Sabuncuoglu and Bayiz 2000; La 2005; Billaut et al. 2007).

Sadeh, Osuka, and Schnelbach, with the system called Micro-Boss, worked on a job-shop scheduling problem where machines may break down (Sadeh et al. 1993). Large Neighborhood Search is used when partially rescheduling, i.e., a set of activities to unschedule (“conflict propagation”) is identified by using recovery rules: all the activities whose start times are changed by these rules are unscheduled. The original scheduling algorithm is then used to reschedule the unscheduled activities. Simple rules are followed for identifying the neighborhood and for conflict propagation:

- Right shift rule: activities are moved later in time while preserving sequence.
- Right shift and jump rule: activities are moved later in time, jumping over ones that do not need to be moved.

[1] Earlier, Muscettola proposed such an approach (Muscettola 2002).

The idea is to use a simple rule to quickly repair each schedule. When fully rescheduling Micro-Boss is used. This system focuses on resource conflicts on a limited time period. Micro-Boss has the ability to detect the emergence of new bottlenecks during the construction of the schedule and change the current scheduling strategy. Micro-Boss is also able to schedule an entire bottleneck (or at least a large chunk of it), i.e., it considers resource conflicts over the complete time horizon. Resource contention is monitored during the construction of a schedule, and the problem-solving effort is constantly redirected towards the most serious bottleneck resource.

During execution, dispatching rules are used to adapt the current schedule. Activities are scheduled one at a time, i.e., every time a machine becomes free, a ranking index is computed for each remaining activity, and the activity with the highest ranking index is then selected to be processed next as follows:

- Weighted Shortest Processing Time (WSPT) tries to reduce overall tardiness by giving priority to short jobs and taking into account the priority of each job.
- Slack per Remaining Processing Time (SRPT) combines the Minimum Slack (MSLACK) rule and the durations of the remaining activities; MSLACK computes the slack time of each activity with respect to the due date of its job and selects the activity with the smallest slack time.
- Weighted Cost OVER Time (WCOVERT) combines WSPT, the expected waiting time of each remaining activity, and MSLACK.
- Apparent Tardiness Cost (ATC) combines WSPT and MSLACK.

According to Sadeh, these heuristics are well suited to solving problems in which some activities have to be performed within non-relaxable time windows as well as repairing schedules in the face of contingencies (Sadeh 1994). Using Micro-Boss is a reasonable, pragmatic approach where there is no explicit reasoning about time to find a solution. The system does not generate stable schedules: the number of activities to reschedule may be large.

Another work of interest about partial rescheduling was done by Smith (1994). OPIS is a full scheduling system based on repeatedly reacting to events. It is a more sophisticated reasoning mechanism for analysis of conflicts than Micro-Boss on the basis of constraint propagation. This approach to incremental management of schedules is based on a view of scheduling as an iterative, constraint-based process.

El Sakkout and Wallace used a different approach to fully reschedule (El Sakkout and Wallace 2000). Their approach consists in a minimal-perturbation rescheduling. Given a schedule and a reduction in resource capacity, their system has to find a new schedule which minimizes the sum of absolute deviations from activity start times in the original schedule. The main idea is to combine Linear Programming (LP) and Constraint Programming (CP), and using potentially good assignments (probes) to solve the problem. The cost function for measuring change involves a kind of earliness/tardiness which is usually well solved by Simplex, if resource constraints are relaxed. One represents temporal constraints and the cost function in a linear program and temporal constraints and resource capacity constraints in CP. One uses the relaxed optimal start times from the linear program to drive the CP branching heuristic, i.e., one adds new linear constraints in the linear program when one branches in the decision tree (CP model) and propagates these decisions. This technique is named “Probe Backtrack Search.” The experiment consists of one (non-unary) resource with a given schedule; when an event reduces resource capacity over some time interval the system reschedules. This method is only practical in situations where the time-to-solve is irrelevant. The optimization criterion of the original schedule is ignored. Their experiments show that this hybrid branch–and–bound technique is better than pure CP, other combinations of CP and LP, and pure Mixed Integer Programming (MIP). These hybrid algorithms are not intended to be used online; to be considered revision solution techniques, the solution approach has to be adapted to take into account the execution of schedules, i.e., unary temporal constraints are added to the problem during execution corresponding to committed activity start and end times.

In AI planning, Wang and Chien (1997) focused on replanning which preserves elements of the original (predictive) plan in order to use more reliable domain knowledge and to facilitate user understanding of produced plans. They presented empirical results documenting the effectiveness of these techniques in a NASA antenna operations application. The authors assume there is a default value for each state goal. They suppose there are well-known methods (activities) for establishing the default value for each state goal. They assume the original plan is applicable from a state where each relevant state goal is at its default value. When an unexpected state change occurs, the replanning algorithm then reuses as much of the original plan as possible while minimizing the amount of re-execution. The replanner returns a plan consisting of the activities that need to be re-executed and those not executed, as well as additional ordering constraints.

3.4 Progressive techniques

The idea behind progressive techniques is to interleave scheduling and execution, by solving the problem piece by piece, where each piece can correspond to the activities in a time window, for example. Reasoning is done as a background task online, i.e., we can afford more time to search, we incrementally commit to scheduling decisions periodically or when new information arrives, and no decisions are changed.

One way of proceeding when using a progressive approach is to select and schedule new subsets of activities to extend the current executing schedule on a gliding time horizon. A decision is made when the uncertainty level of the information has become low enough, and/or when the *anticipation time horizon*, the interval between the current time and the expected end time of the last scheduled activity, has become too small. We can distinguish between the *commitment time horizon*, which is the temporal window in which decisions that are taken should not be reconsidered during execution, and the *reasoning time horizon*, which is the temporal window in which possible actions and expected events are considered for the search process to make relevant decisions. These two time horizons overlap but are not necessary equal. One thus needs an execution-monitoring system able to react and indicate when we have to make new decisions, what information to reason about to make new decisions, and what decisions to make.[2] A short-term schedule on which decisions are made is sometimes called an "overlapping plan" in manufacturing.

Vidal et al. (1996) presented an approach in which they allocate container transfer activities in a harbor to robots only as long as temporal uncertainty remains low enough to be reasonably sure that the chosen robot will actually be the best to choose. The objective is to minimize makespan. There is an anticipation time horizon, but the commitment time horizon merges with the reasoning time horizon.

3.5 Mixed approaches

Interestingly, there are not that many approaches that combine different techniques along the classification that we just proposed. Most of them combine a revision approach with a progressive approach.

In the CASPER planning system (Chien et al. 2000), planning is done on board an autonomous spacecraft, mixing a replanning process when a failure occurs or a science opportunity is observed (i.e., new goal specifications are added to the planning problem), and continuous planning. At each time, the system updates the current plan, goals, state, and next predictive states by using a telescoping-time horizon approach. The current plan is generated given what is expected to happen. When a plan failure happens, an iterative-repair procedure, which is a greedy search algorithm, is applied in the short-term horizon to change allocations, move, renew, or remove actions.

[2]Alternatively, new subsets of activities can simply be integrated periodically, and so no complex conditions are monitored.

Branke and Mattfeld tackled dynamic job-shop scheduling problems making decisions on a gliding time horizon and changing some decisions (Branke and Mattfeld 2002). Note, however, that the progressive approach is not chosen by the authors but required to solve such a dynamic problem in which jobs arrive stochastically over time. No monitoring of execution is used to decide when to select and schedule activities. Activity durations are deterministic. Each job is associated with a due date, and the goal is to minimize the summed tardiness. An evolutionary algorithm is implemented to find schedules with short makespans and low early idle times. The generation–execution approach proceeds as follows. An initial schedule is created with the set of known jobs and the execution of this schedule begins. At some point a new job arrives. All the already executed and executing activities are removed from the scheduling problem, the new job is added and a new solution is found by changing some decisions. When the final activity in a job executes, the contribution to the summed tardiness is calculated. There is no anticipation time horizon, and the commitment time horizon merges with the reasoning time horizon.

Shafaei and Brunn published an empirical study of a number of dispatching rules on the basis of a gliding-time-horizon approach for a dynamic job-shop environment (Shafaei and Brunn 1999a). The first purpose of the study was to find the best dispatching rule, and the second was to investigate the effects of the rescheduling interval on performance and examine whether there is a policy that can always improve performance. Activity durations are imprecise and randomly picked with equiprobability in ranges. The job arrival rate follows a Poisson distribution, and shop load is either heavy or moderate. A release date and a due date are associated with each job. The performance measure considered is an economic objective, which is to minimize the sum of the cost of starting jobs too early and the cost of work-in-progress inventory and tardiness. In general, under tight due-date conditions, the rescheduling interval has a much more significant effect on performance than under loose due-date conditions: the smaller the interval, the lower the costs. There is no anticipation time horizon, and the commitment time horizon merges with the reasoning time horizon.

In another report, Shafaei and Brunn investigated how efficient these dispatching rules are in a dynamic and stochastic shop environment (Shafaei and Brunn 1999b). The number of activities per job is uniformly sampled, job routes are randomly selected, activity durations are imprecise, and machines may break down: machine breakdown intervals and repair times follow exponential probability distributions. The simulation results, under various conditions in a balanced and unbalanced shop, are presented and the effects of the rescheduling interval and operational factors including shop load conditions and a bottleneck on the schedule quality are studied. They conclude that more frequent rescheduling generally improves performance in an uncertain situation. There is no anticipation time horizon, and the commitment time horizon merges with the reasoning time horizon.

There is one approach which, as opposed to the previous ones, does mix proactive (instead of progressive) and revision techniques: the one presented by Bresina et al. (2002), which deals with planning for space rover applications. The authors claim that it is not possible to do timely replanning on board the rover when a failure occurs. They have to deal with uncertainty about the power required, the necessary data storage, the position and orientation of the rover, and environmental factors that influence actions, such as soil characteristics, etc. Therefore, it is necessary to plan in advance for some of the potential contingencies. But the planner can only afford to account for the "important" contingencies, and must leave the rest to run-time replanning. They are also concerned with looking for what we call a *generic* (though flexible here) schedule, through decision-theoretic planning: their objective is to find the plan with the maximum expected utility: each goal has a value or award associated with it, and an action's uncertain effects on continuous variables are characterized by probability distributions.

In his Ph.D. dissertation, Hildum (1994) presented the Dynamic Scheduling System (DSS), an agenda-based blackboard system, which is capable of dealing with a wide range of dynamic Resource-Constrained Scheduling Problems, and producing quality schedules under a variety of real-world conditions. It handles a number of additional domain complexities, such as inter-order tasks (common tasks that are shared by two or more jobs) and mobile resources with significant travel requirements. The solution approach of DSS is a proactive–revision–progression approach, since flexible and stable schedules (with slack time) are generated when a resource brakes down or a new order arrives. Slack time is inserted into the schedules because some resource allocations are still to be done and there are alternative and mobile resources. This solution approach is applied to industrial applications. Furthermore, the basic DSS problem-solving method is essentially a multiple-attribute, dynamic heuristic approach that focuses on the most urgent unsolved subproblem at any point in time. The system can deal with both hard and soft constraints and uses constraint programming techniques.

3.6 Discussion

We can compare the three families of techniques with respect to the following criteria: online memory need, online CPU need, schedule robustness and stability, commitment time horizon, and use of the knowledge about uncertainty.

Revision techniques do not consume a lot of memory online, since we only have to store one schedule. They may require a lot of CPU online, depending on the time spent on rescheduling, which may be done through backtrack search. And, since one will usually not have enough time to process a complete search, the quality of revised solutions may quickly degrade, which means the robustness of schedules is not guaranteed. Stability may be very low, if we change a lot of decisions. Commitment time horizon could be expected to be high, as one computes a global schedule, but it will effectively be very short, since we do not know in advance what decisions will be changed online. The knowledge about uncertainty is not taken into account to make decisions.

Online memory may vary for proactive techniques, depending on whether we have to store one or more schedules: conditional schedules may require a lot of memory. In general, online computational need is low, since we do not have to search online for solutions with backtracking. We can expect to generate highly robust or stable schedules, since we take into account what may occur online to make decisions, but with generic approaches and sometimes with flexible schedules the solution found amounts to a compromise which does not assure getting the highest quality solution in the observed situation. That is not the case with conditional approaches where each branch is expected to be the optimal sub-schedule in the matching situation. The commitment time horizon appears to be long with flexible schedules, but that is at the price of not taking all decisions. In conditional approaches, the commitment time horizon is short as alternatives prevent from being committed to one specific branch.

Progressive techniques permit us to limit our online memory need to the minimum, since we only store a piece of schedule. The requirement in CPU online is balanced, since we need to search but only to solve sub-problems. Decisions are made with a more or less short commitment time horizon and a time granularity. This limited commitment time horizon may prevent from getting very robust schedules, since one loses a global view of the problem to solve, but that actually depends a lot on the type of decisions which are done progressively. If precise scheduling (as in job-shop scheduling) might often be done only in the short term with no loss in the global optimality, that is, of course, not the case with classical planning decisions, where one looks for a causal sequence from the initial state to the final goal state, and hence can hardly do without a global view of the solution. Continuous planning is actually only feasible in applications where new goals arrive periodically and can be treated separately and successively, as in Chien et al. (2000). However, this family of techniques always generates stable schedules.

Table 1 synthesizes the different properties of each family of solution-generation techniques used to handle uncertainty, change, or both for scheduling. These features can help a user to choose a technique in a specific application

Table 1 The properties of each family of solution-generation techniques

	Online memory need	Online CPU need	Quality and robustness	Stability	Commitment time horizon	Knowledge about uncertainty
Revision	Average	High	Low	Low	Very short	No
Proactive generic	Low	Very low	Average	Very high	Long	Yes
Proactive incomplete	Low	Low	High	Very high	Long	Yes
Proactive conditional	Very high	Low	Very high	Very high	Short	Yes
Progressive	Very low	Average	Average	Very high	Short	No

domain: if memory usage is limited, then conditional schedules are probably not the right answer; or if optimality is a key concern, then proactive techniques must be favored, provided that knowledge about uncertainty is available.

Of course, such properties only apply to "pure" proactive, revision, or progressive techniques, and will be qualified somehow when one mixes them: for instance, a progressive conditional approach (i.e., compute several alternatives but with limited commitment time horizon) will need average online memory, with both reasonable online CPU need and rather high robustness. Therefore, one can see how such mixed techniques are interesting for finding better compromises with respect to those features a user wishes to meet in her application. But there are also more pragmatic reasons, making "pure" techniques usually unrealistic: in a highly stochastic world, a pure revision approach would amount to almost permanent rescheduling, which shall be limited, by mixing the revision technique with a proactive or progressive approach (see Sect. 3.5). On the contrary, a pure proactive technique is not always realistic, since there will often be unpredicted or unmodeled deviations that can only be dealt with by a revision technique. And as suggested above, purely conditional global schedules suffer from combinatorial explosion and are often simply infeasible: developing only some of the branches progressively is a way to overcome it.

As a matter of conclusion, a user should be given a global system encompassing the three kinds of approaches, allowing her to tune the levels of proactivity, progression, and revision that will best fit her needs. As we have seen, few mixed techniques have been proposed for making scheduling decisions in a stochastic environment, but as far as we know, no one has proposed a system or an approach that combines the three ways of scheduling under uncertainty.

4 Representation

This section describes a model for scheduling in a stochastic execution environment. This model integrates the three families of approaches presented in the previous section. Our model is not intended to be formal on each aspect of the scheduling problem, but only on the key and new features related to when and how decisions are made: in order to also remain as generic as possible with respect to different problem formulations and search techniques existing in related literature, the model will be less formal on less relevant aspects (as, for instance, the representation of resource constraints).

4.1 Schedule

We are interested in extended scheduling problems with mutually exclusive subsets of activities, in a way similar to what was done by Tsamardinos et al. (2003) and by Beck and Fox (2000). At the roots of our model, we need variables and constraints inspired by the constraint paradigm.

Definition 4.1 (Variable) A variable is associated with a domain of values or symbols, and it is instantiated with one and only one of the values or symbols in this domain.

Definition 4.2 (Constraint) A constraint is a Boolean function relating one (unary constraint), two (binary constraint), or more (k-ary constraint) variables that restrict the values that these variables can take.

The domain of a variable is reduced by removing values that can be proved to not take part of any solution (given the decisions already made).

We distinguish between two types of variables in the problem: the *controllable variables* and the *contingent variables*.[3]

Definition 4.3 (Controllable variable) A controllable variable is a variable instantiated by a decision agent.

One of the issues we are interested in this paper, with respect to controllable variables, is to decide when to instantiate them. For example, it may be difficult to set activity start times in advance when activity durations are imprecise because of temporal uncertainty.

Definition 4.4 (Contingent variable) A contingent variable is a variable instantiated by Nature.

Moreover, a (probabilistic/possibilistic/etc.) distribution of possible values may be attached to each contingent variable, when knowledge about uncertainty is available and one wishes to use it. Such distributions are updated during execution: they can be revised when new information arises, or they can be truncated when one has observed, for example, that an activity with an imprecise duration has not yet finished.

We can now define the basic objects of a scheduling problem, namely resources and activities.

Definition 4.5 (Resource) A resource r is associated with one or more variables, that represent its capacity, efficiency, and/or state. Its capacity is the maximal amount that it can contain or accommodate at one time point. Its efficiency describes how fast or how much it can do with respect to its available capacity. Its state describes its physical condition. A resource capacity, efficiency, and state can all vary over time. These variables are either controllable or contingent. r is related to a global resource constraint ct_r on all its variables and the variables of the activities that require it. The scheduling problem comprises a finite set of resources noted $\mathcal{R}$.

We can model the state of the execution environment as a set of state resources, e.g., the outside temperature is modeled by a resource that can be in only one of three states depending on time: hot, mild, and cool.

Definition 4.6 (Activity) An activity $a = \langle start_a, d_a, end_a, [\mathcal{CT}_a]\rangle$ is defined by three variables: a start time variable $start_a$, a duration variable d_a, and an end time variable end_a. These variables are either controllable or contingent. a may be associated with an optional set of resource constraints $\mathcal{CT}_a$ that involve the variables of the resources it requires.

In a constraint-based model, we usually post the following constraint for each activity: $end_a - start_a \geq d_a$. Of course, constraints of any type between variables can be posted on our scheduling problem, e.g., we can post temporal constraints.

We assume that the distributions of possible values for contingent variables are independent from the values chosen for controllable variables, except for the end-time variables of the activities that have contingent duration variables. The activity end-time variables are contingent, and their distributions depend on the start times chosen for these activities. For example, when we choose the resource to allocate to an activity with a probabilistic duration, this does not effect the probability distribution associated with the activity duration.

Our scheduling problem is composed of resources, activities, and constraints relating them, with possibly additional variables describing the state of execution environment.

To fit the classification described in Sect. 3, additional constraints may have to be posted by the schedule-generation algorithm to set resource allocations, make sequencing decisions, and/or set precise activity start times. For example, when we want to set the start time of an activity, we post a new unary temporal constraint that relates to the activity start time variable.

Central to our model is the notion of *conditions* that are subsets of variables related by logical and/or arithmetic relations: such conditions guide the branching within conditional schedules, the selection of new subsets of activities in a progressive technique, etc.

Definition 4.7 (Condition) A condition $cond = \langle func, [atw]\rangle$ is a logical and/or arithmetic relation *func* in which at least one variable is involved. It may be associated with an optional active temporal window that is an interval $atw = [st, et]$ between two time-points st and et in the current schedule. If $st = et$, then it means the condition must be checked at a precise time-point in the schedule.

A condition can involve characteristics of the distributions of contingent variables. A condition can be expressed with conjunctions and disjunctions of conditions.

A typical example of a condition is what we will call a *branching condition*. A branching condition will be attached to one of mutually exclusive subsets of activities (see below) and will be checked at a specific time point that we will call a *branching node*. For example, given three mutually exclusive subsets of activities and a contingent variable representing the outside temperature, we observe the value of this contingent variable at the start time of activity a_{11} (see Fig. 1). The three branching conditions are the following:

[3]Controllable variables correspond to decision variables, and contingent variables to state variables in the Mixed Constraint-Satisfaction Problem framework (Fargier et al. 1996).

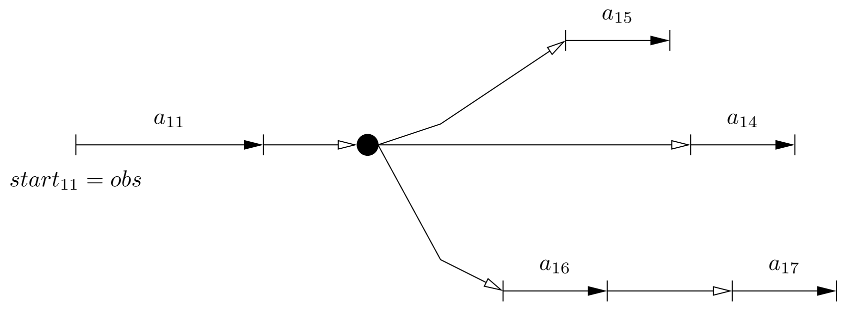


Fig. 1 Example of a branching condition with three mutually exclusive subsets of activities

temperature is below 20 degrees, temperature is between 20 and 30 degrees, and temperature is above 30 degrees. When the first condition is met, then we choose a_{15}; when the second condition is met, then we choose a_{14}; when the third condition is met, then we choose a_{16} and a_{17}.

We propose the following recursive definition of a schedule to describe our model with respect to these particular mutually exclusive subsets of activities.[4]

Definition 4.8 (Schedule) A schedule $\mathcal{S}$ is either
• void $\mathcal{S} = \emptyset$, or
• $\mathcal{S} = \langle a_\mathcal{S}, \{\mathcal{CT}_\mathcal{S}\}^*, \mathcal{S}'\rangle$ is an activity $a_\mathcal{S}$ partially ordered via constraints in $\{\mathcal{CT}_\mathcal{S}\}^*$ with respect to the activities of a schedule $\mathcal{S}'$, or
• $\mathcal{S} = \langle bnd_\mathcal{S}, nb_\mathcal{S}, \{rcp_\mathcal{S}\}^*, cnd_\mathcal{S}\rangle$ is a set of $nb_\mathcal{S}$ mutually exclusive recipes $rcp_\mathcal{S}$, i.e., mutually exclusive recipes represent different ways of attaining the same goal, as defined below; such recipes follow a branching node $bnd_\mathcal{S}$ and lead to a converging node $cnd_\mathcal{S}$, i.e., there are precedence constraints that enforce these ordering relations. A node is a dummy activity a^{dum} of null duration that does not require any resource: $a^{\text{dum}} = \langle start_a^{\text{dum}}, 0, end_a^{\text{dum}}\rangle$, with $start_a^{\text{dum}} = end_a^{\text{dum}}$.

Definition 4.9 (Recipe) A recipe $rcp = \langle \mathcal{S}, [Py_{rcp}], bc_{rcp}\rangle$ consists of a schedule $\mathcal{S}$ associated with an optional probability, possibility, or plausibility of being executed Py_{rcp} and a branching condition bc_{rcp}: it will be executed if and only if bc_{rcp} is met.

A recipe can describe one of several possibilities for performing an action, e.g., a device can be built in different ways that are mutually exclusive. At execution, for each set of mutually exclusive recipes, only one will be executed.

The first two ways of defining a schedule are just an alternative way to recursively define a classical partially ordered schedule without alternatives. The third introduces parts of a schedule that divide, at some given branching node, into mutually exclusive recipes: each recipe rcp_i will be executed, if a branching condition is met at that point.

It should be noted that conditions must be designed such that they are actually mutually exclusive and cover all cases. Moreover, as usual with probability distributions, if ever they are used, the sum of probabilities associated to the $nb_\mathcal{S}$ branches must equal 1.

The previous recursive definitions permit us to build a schedule piece by piece, building subsets of partially ordered activities that are then composed into a set of mutually exclusive recipes, this set being in turn integrated into a subset of partially ordered activities that is in turn one of several mutually exclusive recipes, and so on: alternatives may be nested within alternatives.

For tractability reasons, we assume there are no temporal constraints between two activities that do not belong to the same recipe. However, some precedence constraints must be added to constrain branching conditions to be checked before their related recipes would be executed.

4.2 Generation and execution

We defined a model that a proactive method could use to generate a schedule that would then be entirely sent to the execution controller. To make it possible to use revision and progressive techniques, we need to consider a situation in which a schedule is executed in a highly stochastic environment, thus requiring scheduling decisions to be made and/or changed while executing. Hence, we need to design a model interleaving schedule generation and execution: the resulting system must be able to react, to know what to do (e.g., reschedule, schedule next subset, make new scheduling decisions, etc.), and to know how to do it.

Two types of algorithms will be needed: *execution algorithms* will be in charge of dealing with the current schedule and both making the scheduling decisions that remain and actually executing activities; *generation algorithms* will be in charge of changing the current schedule, either because some part is not valid anymore and must be modified (revision approach), or because new activities must be added (progressive approach).

[4]One should notice that what we call a schedule would be better referred to as a *solution* to a scheduling problem, which is possibly not fully set but only defines a partial order. In the Operations Research community, a schedule implies that all activity start times are set.

The dynamic evolution of our model will be monitored via condition meeting: if such a condition is met, then we know we have to make or change decisions. A branching condition is actually used by the execution algorithms, guiding them into the proper alternative. We need to introduce here two new types of condition: a *generation condition*, when met, activates a new generation step through the generation algorithm; then a *fire condition* will actually enforce the global monitoring system to turn to this newly generated schedule. Such generation and fire conditions are needed both in revision and progressive approaches.

Typical examples of generation and fire conditions are violations of some constraints in the current schedule, arrivals of new activities to execute, critical resources no longer available (implying a revision mechanism), or the anticipation time horizon has become too small and so we need to schedule a new subset of activities to anticipate execution (implying a progressive mechanism).

The generation and execution model can be represented by a dynamic directed acyclic graph (DAG), similar to a finite-state machine, whose nodes are called execution contexts.

Definition 4.10 (Execution context) An execution context $ect\langle \mathcal{S}_{ect}, \alpha_{ect}\rangle$ is composed of a schedule $\mathcal{S}_{ect}$ and an execution algorithm α_{ect}.

$\mathcal{S}_{ect}$ is a flexible or complete schedule. In addition, it is possible for an execution context to not contain all recipes starting from a branching node, but only those with the highest values Py; another example of generation condition is hence that when the value Py of a remaining recipe becomes high enough, we generate in a progressive way a new schedule composed of the current schedule and a new recipe.

α_{ect} makes decisions (start time setting, resource allocations, branching on one recipe among several candidates, etc.) greedily: it cannot change decisions that are already made. In case of pure execution approach, such as dispatching, α_{ect} makes all decisions.

The arcs of our dynamic DAG are transitions between execution contexts.

Definition 4.11 (Transition) A transition $tr = \langle ect_{tr}^{src}, ect_{tr}^{tat}, cond_{tr}^{\mathrm{gen}}, cond_{tr}^{\mathrm{fir}}, \beta_{tr}\rangle$ is composed of a source execution context ect_{tr}^{src}, a target execution context ect_{tr}^{tat}, a generation condition $cond_{tr}^{\mathrm{gen}}$, a fire condition $cond_{tr}^{\mathrm{fir}}$, and a generation algorithm β_{tr}.

The default situation for the temporal windows of the generation and fire conditions of transition tr is the whole source execution context ect_{tr}^{src}, i.e., their temporal windows equal the maximal interval between the start point and the end points of ect_{tr}^{src}.

When generation condition $cond_{tr}^{\mathrm{gen}}$ is met, generation algorithm β_{tr} generates target execution context ect_{tr}^{tat} from source execution context ect_{tr}^{src} and the data of the problem model. Execution algorithm $\alpha_{ect_{tr}^{tat}}$ is set by β_{tr} from a library of template execution algorithms, that depend upon the degree of flexibility (incompleteness or branching) of the generated schedule. β_{tr} can decide or change a part of or all decisions; in particular, it can select a subset of activities to include into ect_{tr}^{tat} (progressive approach). Transition tr is fired when its fire condition is met. When tr is fired, we change contexts: we go from source execution context ect_{tr}^{src} to target execution context ect_{tr}^{tat}. Generation condition $cond_{tr}^{\mathrm{gen}}$ must be more general than or equal to fire condition $cond_{tr}^{\mathrm{fir}}$ since $cond_{tr}^{\mathrm{gen}}$ must be met before or when $cond_{tr}^{\mathrm{fir}}$ is met.

Template transitions are defined offline, and each of them is an implicit description of many transitions that may appear in a model. A template transition is active when it can be instantiated, i.e., it is active when one of its implicit transitions can generate a new context. For example, a template revision transition associated with a resource constraint rct_1 is active each time one of the activities involved in rct_1 is executing and allocated to the resource involved in rct_1: when the associated resource breaks down, then we have to change a number of allocation decisions.

The generation algorithm generating ect_{tr_l} and the execution algorithm associated with ect_{tr_l} are complementary: the former makes some decisions for a subset of activities, and the latter makes the remaining decisions for these activities, e.g., the former makes allocation and sequencing decisions, and the latter sets activity start times.

It should also be noted that all conditions are checked by execution algorithms. When a branching condition is met, we do not change execution contexts. When a generation condition is met, a new execution context is generated. When a fire condition is met, we change execution contexts.

Our first assumption is that uncertainty level decreases when executing a context. Ergo, we leave some decisions to the execution algorithm to limit the computational effort that would be used to revise decisions, and the perturbations and instability due to such revision. Decisions that can be made in advance because they concern variables with low uncertainty are taken by generation algorithms, while remaining decisions will be taken later either by generation or execution algorithms when their uncertainty will be lower.

Our second assumption is that dynamics of the underlying controlled physical system are low enough with respect to the time allotted to the reasoning system to search for schedules online. Therefore, one has enough time to generate new contexts before they are executed. Generation algorithms should be anytime, i.e., generation algorithms should be able to produce a schedule whose quality, robustness, or stability increases with search time. In principle, the decisions made by generation algorithms are better with respect to an optimization criterion than the decisions made by execution algorithms. The former have more time to reason and choose the best schedules among a set of executable schedules, whereas the latter are greedy and return the first executable they find. Generating a new execution context and switching into it are not necessary synchronous, this permits us to generate new execution contexts during the time period in which the current execution context is under control and executing.

One can see that any 'pure' technique can be easily instantiated with our model. A pure proactive technique will only need a single context, generation being made once offline, the remaining decisions being taken by the sole execution algorithm. In a pure revision approach, contexts contain complete predictive schedules with basic execution algorithms, and generation/fire conditions associated with failures or quality deviations in the current context, and generation algorithms change the current schedule to fit the new situation. In a pure progressive approach, contexts contain complete schedule pieces with standard execution algorithms, and generation/fire conditions related to the anticipation time horizon getting too small or the uncertainty level decreasing, and generation algorithms change the current schedule by inserting a new subset of activities into it. The strength of the model is that now all three kinds of approaches can be integrated and parameters can be tuned to put more or less flexibility, more or less revision capabilities, etc., upon needs that are driven by the application.

4.3 An illustrative example

This section presents an illustrative example in the context of the construction of a dam that shows how our generation–execution model is applied in practice.

We assume we have to schedule a set of activities:

- Construction of three roads $road_1$, $road_2$, and $road_3$
- Construction of two houses $house_1$ and $house_2$
- Digging of a tunnel in one of two alternative ways $tunnel_1$ and $tunnel_2$ depending on geological conditions observed at the end of $road_1$
- Preparation of dam foundations *foundation*

In addition, we assume we can use two trucks, and each activity requires one of them except $house_1$ that requires both. There are precedence relations between activities, due dates, and a tardiness-cost function. Activity durations are imprecise and modeled by probability distributions. Our objective is to generate a schedule that minimizes the expected tardiness cost.

Figure 2 represents the temporal constraints between activities of this small dam construction project. This is a directed AND/OR graph. Each box symbolizes an activity, and the box lengths depend on activity durations. Arcs represent precedence constraints between activities, e.g., $road_2$ must start after the end of $road_1$. Conjunctions are represented by arcs connected with small arcs of a circle, e.g., activities $house_2$ and $road_2$ might be executed in any order or in parallel. Disjunctions are represented by arcs that are not connected with arcs of a circle, e.g., activities $tunnel_1$ and $tunnel_2$ are mutually exclusive. Each recipe is associated with a probability of being executed that may vary during execution.

When creating a generation–execution model for this illustrative problem, the first phase consists in deciding whether we want to monitor schedule execution. Since our knowledge about uncertainty is limited to probability distributions, we want to be able to change decisions when expected tardiness cost deviates too much from its predictive value. In addition, we want to be able to make decisions on a gliding time horizon to limit our commitment. For these reasons, we need to monitor schedule execution. We decide to design two template transitions. The revision transition is active during execution, whereas the progression transition is active as long as there is at least one activity that is still not scheduled. We decide to use the same execution algorithm for any execution context. This execution algorithm fixes activity start times as early as possible.

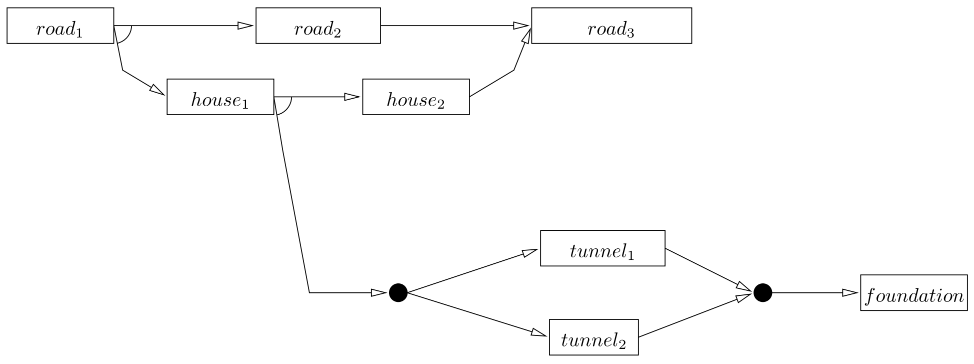


Fig. 2 Temporal representation of a small dam construction project

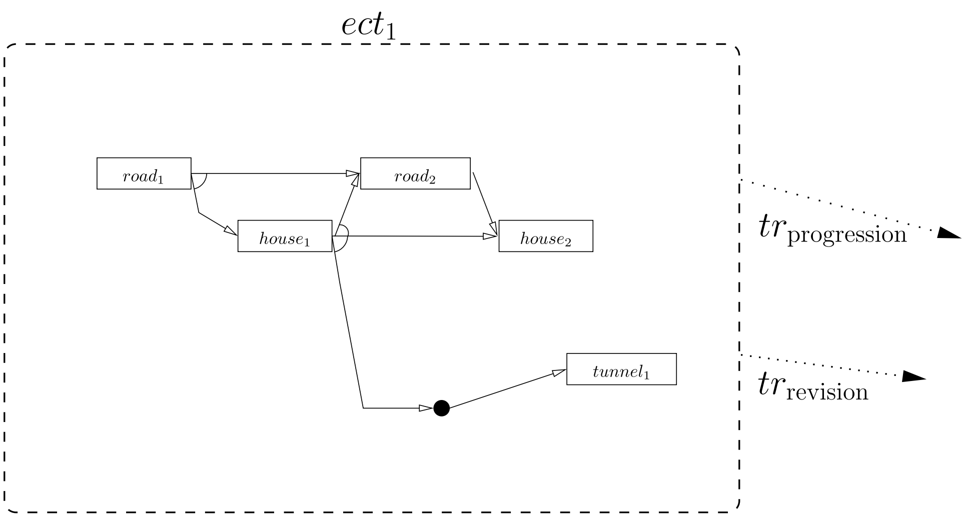


Fig. 3 Execution context generated before starting execution

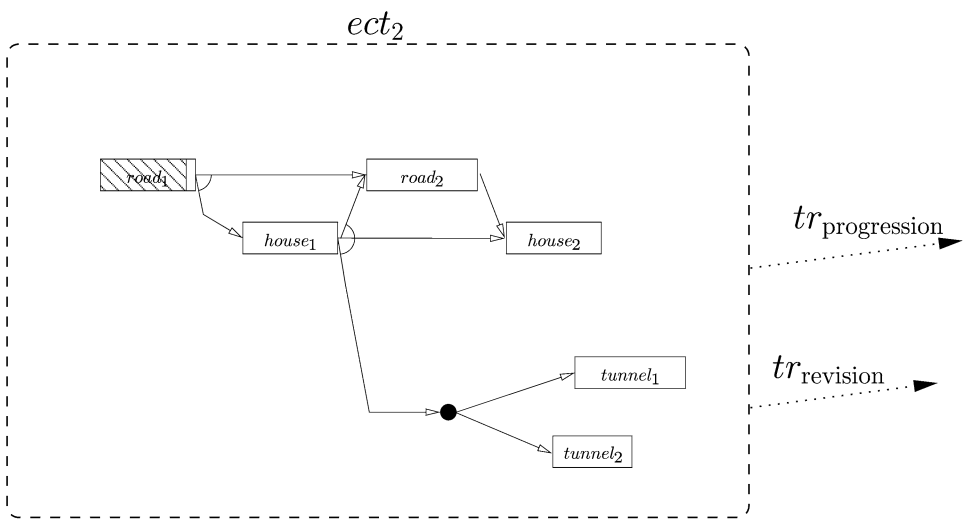


Fig. 4 Execution context generated during the execution of *road*$_1$

The first execution context *ect*$_1$ is generated offline since we need to schedule at least one activity before starting execution. Figure 3 represents *ect*$_1$. Notice that *ect*$_1$ only includes a subset of activities since our progression generation algorithm makes scheduling decisions on a short-term horizon. *tunnel*$_1$ is scheduled because its probability of being executed is much higher than the one associated with *tunnel*$_2$ at this time. Two additional precedence constraints are included in *ect*$_1$ as we cannot use more than two trucks at the same time: *road*$_2$ starts after the end of *house*$_1$; *house*$_2$ starts after the end of *road*$_2$. These sequencing decisions are made to minimize the expected tardiness cost given probability distributions.

During execution we monitor the probabilities associated with recipes. We start executing *ect*$_1$. During the execution of *road*$_1$, we assume the probability associated with *tunnel*$_1$ decreases, so the probability associated with *tunnel*$_2$ increases. These changes fire the progression transition, and a new context *ect*$_2$ is generated. Figure 4 represents *ect*$_2$ that includes both *tunnel*$_1$ and *tunnel*$_2$.

When *road*$_1$ finishes, we observe geological conditions and choose *tunnel*$_2$ accordingly. Figure 5 represents *ect*$_2$ at this time. Note that the branching node disappears in *ect*$_2$.

The execution of *ect*$_2$ continues until we observe that the mean end time of *house*$_1$ is later than predicted in such a way that *house*$_2$ is delayed and the expected tardiness cost increases too much with respect to its predicted value. This fires the revision transition that creates a new context *ect*$_3$. Figure 6 represents *ect*$_3$, where *road*$_2$ is now sequenced after *house*$_2$.

We are now executing *ect*$_3$. When *house*$_2$ is executing, the progression transition is activated since we need to select and schedule new activities to maintain the lead of generation over execution. A new context *ect*$_4$, which includes all activities, is generated, see Fig. 7. Notice that the template progression transition is no longer active as all activities have now been scheduled.

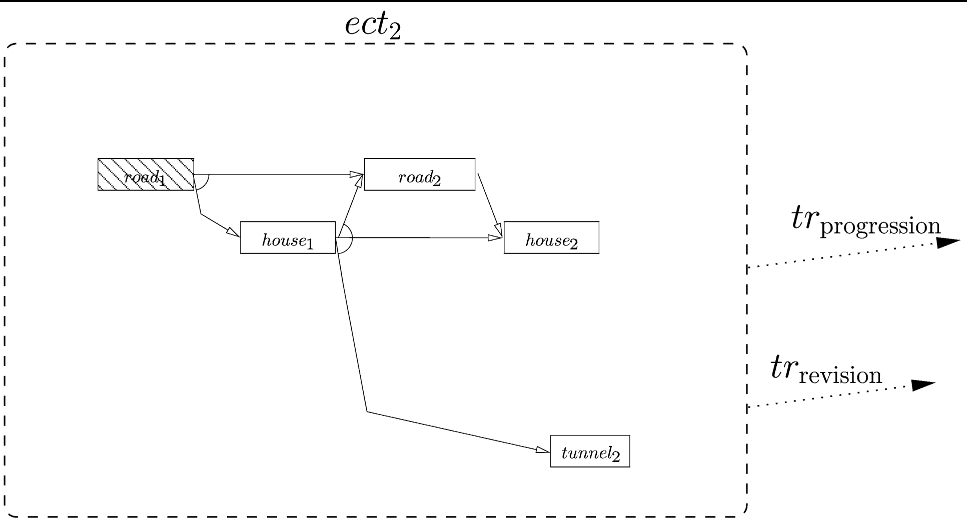


Fig. 5 Execution context generated when *road*$_1$ ends

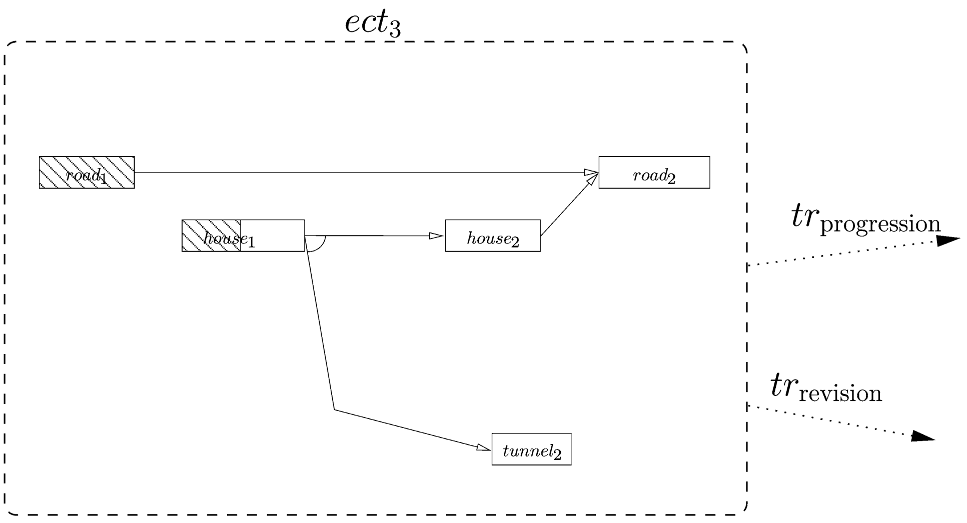


Fig. 6 Execution context generated during the execution of *house*$_1$

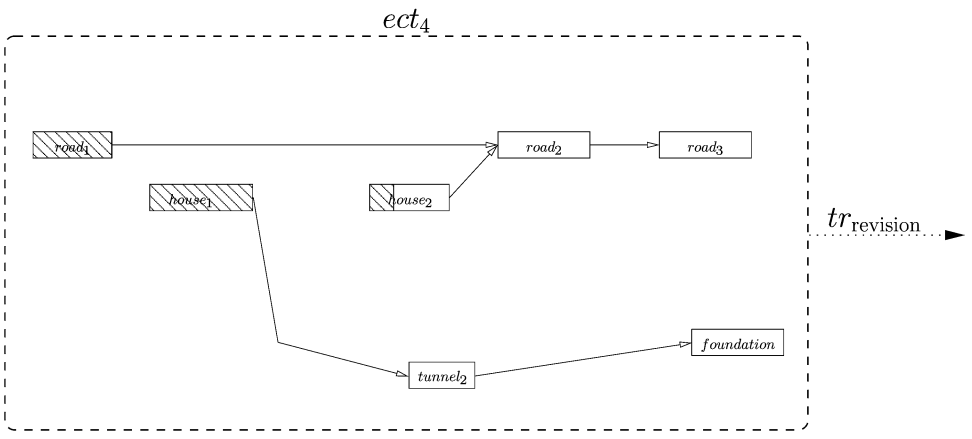


Fig. 7 Execution context generated during the execution of *house*$_2$

5 Experimental system

In this section, we present a few software prototypes that we implemented, and we show how they are special cases of our global model and hence validate it. Detailed experimental results using these prototypes appear in the cited papers.

5.1 Scheduling problem

The *flexible job-shop scheduling problem* (flexible JSSP) is a scheduling problem where the set of activities A is partitioned into *jobs*, and with each job a total ordering over a subset of the activities of A is associated. Each activity specifies a set of alternative resources. An activity must be assigned to one of its alternative resources and execute without interruption on the assigned resource. No activities that are assigned to the same resource can overlap their executions. We represent these non-overlapping activities formally by a partition of A into *resource sets*. The partition of A into resource sets is based on which resource each activity is assigned to, i.e., resource sets are made during the search.

A *solution* corresponds to allocating one resource to each activity and a total ordering on each resource set such that the union of the resource and job orderings is an acyclic relation on A.

For our experimental investigations, we focused on probabilistic flexible job-shop problems. Activity end times are observed during execution. Moreover, resources may break down: for each resource, the duration between two consecutive breakdowns and breakdown durations are observed during execution as well. Random variables are fully independent and associated with probability distributions. Random variables represent activity durations, durations between breakdowns, and breakdown durations. We focused our experimental study on two criteria to minimize: makespan, and sum of tardiness and allocation costs.

5.1.1 *Tardiness cost*

Each job $job_i \in JOB$ is associated with a due date due_i. If the last executed activity of job_i finishes later than due_i, then a *tardiness cost* is incurred:

$$tardiCost_i^{\mathrm{eff}} = \phi_i \times \max(endJob_i^{\mathrm{eff}} - due_i, 0),$$

where ϕ_i is a positive weight, and $endJob_i^{\mathrm{eff}}$ is the effective end time of job_i, observed during execution. A schedule is associated with a tardiness cost $K_{\mathrm{tardiness}}$ defined as follows:

$$K_{\mathrm{tardiness}} = \sum_{\forall job_i \in JOB} tardiCost_i^{\mathrm{eff}}.$$

5.1.2 *Allocation cost*

Another cost, called *allocation cost*, is associated with each resource allocation: when a given activity $a_{ij} \in A$ is executed with a given resource $r_l \in R_{ij}$, it incurs a cost $allocCost_{ijl}$. A schedule is defined by a set of allocations and associated with an allocation cost $K_{\mathrm{allocation}}$: each activity a_{ij} is effectively allocated to a resource $r^{\mathrm{eff}} \in R_{ij}$ and this allocation costs $allocCost_{ij}^{\mathrm{eff}}$, so

$$K_{\mathrm{allocation}} = \sum_{\forall a_{ij} \in A} allocCost_{ij}^{\mathrm{eff}}.$$

5.1.3 *Global cost*

The global cost K represents the whole cost of a solution; this cost takes both allocation and tardiness costs into account. It is formally defined as:

$$K = K_{\mathrm{allocation}} + K_{\mathrm{tardiness}}.$$

Each scheduling problem is characterized by a maximal global cost, $K^{\max}$, that defines an upper bound with respect to the schedule costs. We want to maximize the probability that the global cost is less than the maximal global cost (robustness), i.e., we want to maximize the level of service of each schedule; this can be formally expressed as follows:

$$\max \Pr(K \leq K^{\max}).$$

Note that tardiness and allocation costs are antagonistic. The tardiness cost is likely to be high, if we only want to reduce allocation cost by choosing systematically the cheapest resources when allocating activities so that a small number of activities can be scheduled in parallel. Conversely, the allocation cost is likely to be high, if we only want to reduce tardiness cost by choosing systematically the most expensive resources when allocating activities so that a large number of activities can be scheduled in parallel.

5.2 Architecture

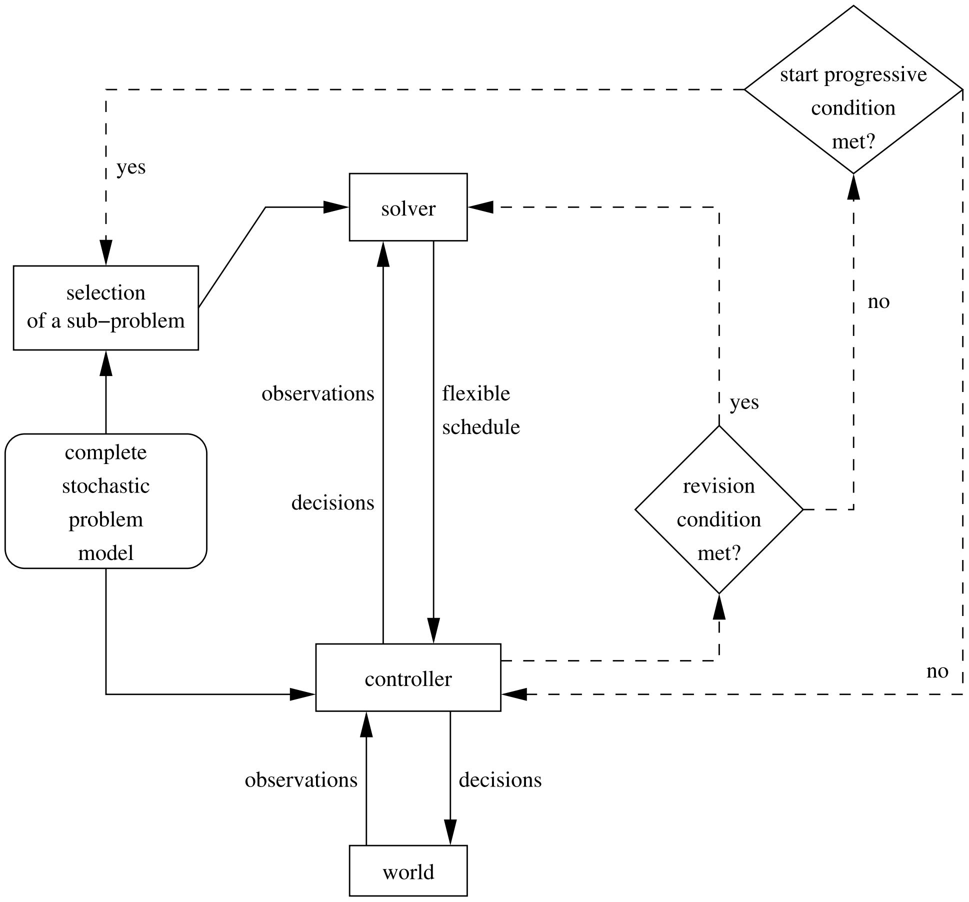
Our experimental system is composed of the following modules: a solver, a controller, and a world.[5] Figure 8 represents our software architecture. Plain arcs represent data flows, while dotted arcs represent data controls. Boxes represent architecture modules.

A solver module is in charge of making decisions with a backtrack search and constraint propagation. The input of the solver module is a stochastic model with variables and conditions. The resolution procedure first computes a deterministic approximation of this model, i.e., we pick a value for each random variable given its probability distribution. The computation can be parametrized, see Sect. 5.5.2. Then, we partially explore the search tree corresponding to the optimization of this deterministic model in depth-first search. The search procedure sequences the activities on each resource, and thus produces flexible solutions because activity start times are not set. We search for flexible solutions until the allotted time for searching is elapsed, i.e., we use an anytime algorithm. When we stop the resolution procedure the best flexible schedule is sent to the controller module.

The controller module is responsible for choosing activity start times given decisions made by the solver module

[5]The world module is not a real execution environment but a simulator of it, it instantiates random variables.

Fig. 8 Architecture of our experimental system



and what happens during execution (observations sent by the world module). The controller module monitors progression and revision conditions to start either a selection of new activities or a re-optimization, when it is relevant. The controller module is also in charge of maintaining known and unknown probability distributions online. For updating unknown probability distributions, associated with contingent variables, such as activity end times, we use Monte Carlo simulation. For computing expected activity end times, we run a series of simulations on the flexible schedule at hand by assuming activities are executed as soon as possible. For each activity a_{ij} that is allocated and sequenced but not yet executed, we randomly pick a set of possible durations. The probability distributions we know, such as the probability distributions associated with activity durations, are truncated and normalized. Suppose we have an activity whose execution started at date $t = 0$ and whose duration, which is between 10 and 60, follows a normal law of mean 30 and standard deviation 10. This activity duration is depicted by the dotted curve in Fig. 9. Now suppose that at date $t = 35$ the activity is still executing, we will assume that the probability distribution of the activity end time is the initial law truncated on the interval [35, 60] and renormalized as shown by the solid curve in Fig. 9. More formally, if the initial

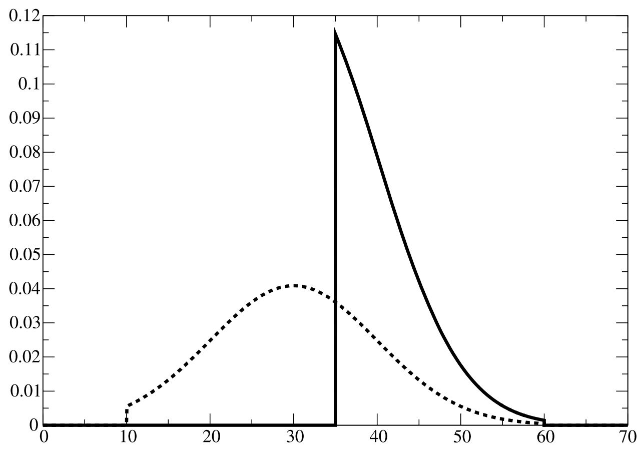


Fig. 9 Two truncated and normalized normal laws

probability law of an activity duration is described by a distribution function $p_0(dur)$ defined in $[dur^{\min}, dur^{\max}]$, and if the activity has been executed $dur0$ units of time, the current distribution is defined by the probability distribution $p_{dur0}(dur)$ as follows:

$$\forall dur \in \left[dur0, dur^{\max}\right], \quad p_{dur0}(dur) = \frac{p_0(dur)}{\int_{dur0}^{dur^{\max}} p_0(t)\,dt}.$$

A Monte Carlo simulation run consists in randomly picking a value for each random variable such that the two sets of values that are randomly picked for two consecutive simulation runs may be completely different.

For computing expected activity end times, we run a series of simulations on the flexible schedule at hand by assuming activities are executed as soon as possible. For each activity a_{ij} that is allocated and sequenced but not yet executed, we randomly pick a set of possible durations, and for each resource that is allocated to a_{ij}, we randomly pick a series of breakdowns. For generating resource breakdowns, we proceed as follows: for each resource allocated to a_{ij}, the last breakdown generated occurs after the end time of a_{ij}; if a breakdown overlaps with the activity, then the activity end time is delayed. For running simulation we use a precedence graph that is generated for the temporal constraint network: each node represents an activity, and each arc represents a precedence constraint between two activities. We then topologically sort the precedence graph and use this ordering in the simulations. The simulation horizon equals the sum of all activity durations. The complexity of a single simulation run is in $O(nbBk + nbAct + nbPCt)$, where $nbBk$ is the number of breakdowns, $nbAct$ is the number of activities, and $nbPCt$ is the number of precedence constraints; it is in $O(nbBk + nbAct)$ for our problem, since there are fewer unary resources than activities and our problem has the same number of precedence constraints as a job-shop scheduling problem.

When progression conditions are met, we have to select and schedule a new subset of activities to anticipate execution, e.g., when the time period between the current time and the end time of the last scheduled activity is too short, we need to select a new set of activities. This permits progressive decision-making since the whole scheduling problem is split into sub-problems and solved piece by piece during execution with uncertainty level decreasing. We have to change scheduling decisions, i.e., we have to revise the current schedule when revision conditions are met: the current schedule is no longer executable or its expected quality deviates too much with respect to its baseline quality computed just after its generation by the solver module. For example, we reschedule when a resource breaks down. The generation–execution loop is controlled by the controller module on the basis of these conditions.

5.3 Revision approach

Our experimental revision approach is parametrized by choosing a *revision criterion* and a *sensitivity factor*. A revision criterion is a condition that is monitored during execution. For example, we monitor the absolute difference between the expected quality, computed before execution, and the current expected quality, computed during execution based on what we observe. We compare this absolute difference with a reference value, and if the revision criterion is met, then we reschedule. A sensitivity factor sets the sensitivity of the revision criterion with respect to perturbations that occur during execution. The sensitivity factor is set to indirectly choose the search effort that depends on the number of reschedulings that occur online (Bidot et al. 2003).

5.3.1 Experimental results

In this section, we report the results of two experimental studies aiming at determining the impacts of the revision criterion and the uncertainty level on makespan with a limited computational effort. We address JSSPs with probabilistic durations, and we assume resources cannot break down. All the programs run to perform these tests have been written in C++, use ILOG Scheduler 5.3 (ILOG 2002) for scheduling, and have been compiled with Microsoft Visual C++ 6. The results presented here have been obtained with using a dual processor Pentium II Xeon 450 MHz with 256 MB.

Impact of revision criterion on makespan We investigated three different revision criteria for deciding when to reschedule.[6]

First, we describe formally the three criteria. All these criteria depend on a strictly positive parameter called the sensitivity factor ω that can tune the sensitivity of the revision criteria. When ω is very small, there will not be any rescheduling. The larger the sensitivity factor, the higher the number of reschedulings (and the smaller the stability of the schedule).

The first criterion revCrit_1 consists in monitoring the makespan and is defined as follows: we reschedule when

$$M_{\text{exp}} > \frac{M_{\text{pre}}}{\omega},$$

where M_{exp} is the current expected makespan, and M_{pre} is the makespan of the current predictive schedule. This criterion will not result in rescheduling, if the effective activity durations are shorter than expected. In other words, it does not allow "opportunistic" rescheduling that would take advantage of unexpectedly short execution times.

A second variation revCrit_2, based on the first, is opportunistic because it reschedules based on the absolute difference between the expected makespan and the predictive makespan. We reschedule when

$$|M_{\text{exp}} - M_{\text{pre}}| > \frac{D}{\omega},$$

where D is the mean of all activity durations of the initial problem.

[6]The work reported here was partially published (Bidot et al. 2003).

The two versions based on makespan monitoring are a priori rather crude: we mainly take into account the observed durations of the activities of the critical paths. If these activities do not slip too much, then, as a consequence, the makespan will not slip too much either. However, it is possible that the expected makespan remains approximately the same while the executions of the activities that do not belong to the critical paths are such that it is possible to find a much better solution by rescheduling. We thus propose a third variation of the approach revCrit$_3$ that takes into account each activity duration. We reschedule when

$$\frac{\sum_{A_{\text{new}}} |end_{\text{exp}} - end_{\text{pre}}|}{nbNewAct} > \frac{D}{\omega},$$

where A_{new} is the set of *nbNewAct* activities that were executing the last time we rescheduled, and end_{exp} and end_{pre} are the expected and predictive activity end times, respectively.

The problem instances with imprecise durations are generated from classical JSSP instances (Lawrence 1984; Applegate and Cook 1991; Adams et al. 1988). Each activity is associated with a normal probability distribution with mean, p, corresponding to the duration in the classical instance, and with standard deviation $\alpha \times p$ with $\alpha > 0$. α characterizes the degree of uncertainty of the problem. The higher α, the larger the standard deviation of each activity duration so the more uncertainty in the system. α is constant and equals 0.3 for each experiment done in this first study.

During schedule execution, whenever we are informed by the environment that an activity ends, we update all our data structures, and simulate the continued execution of the schedule. The updating frequency is sufficiently low to permit 1,000 simulation runs each time. When the end of an activity is observed, there might be other concurrent activities still executing for which distributions are updated. We use simulation to calculate the revision criterion, and then we reschedule only if the revision criterion is met.

Scheduling and rescheduling are done using the constraint-programming approach with a standard chronological backtracking algorithm and a time limit of one second. The new schedule is the best solution found within the time limit using a depth-first search. Two heuristic rules are used to make sequencing decisions: first, we have to select the resource and second, we have to select the activity to rank first on this resource. The resource r with the minimal global slack is selected with priority, and then the activity requiring r, with the minimal latest start time is ranked first on r. The global slack of a resource is the slack time of the activities that are allocated to it. The global slack is computed on the basis of the temporal window of each activity, i.e., this temporal window is the time period between its earliest start time and its latest end time. During search, constraint propagation is used to prune the search tree, in particular Edge-Finder (Nuijten 1994; Baptiste and Le Pape 1996). Note that we do not use simulation to filter solutions found by the tree search because we assume that a solution with a low deterministic makespan is also a solution with a low expected makespan (Beck and Wilson 2007).

The results shown on Fig. 10 come from running 100 different execution scenarios per value of the sensitivity factor ω. These results are obtained from experimenting with

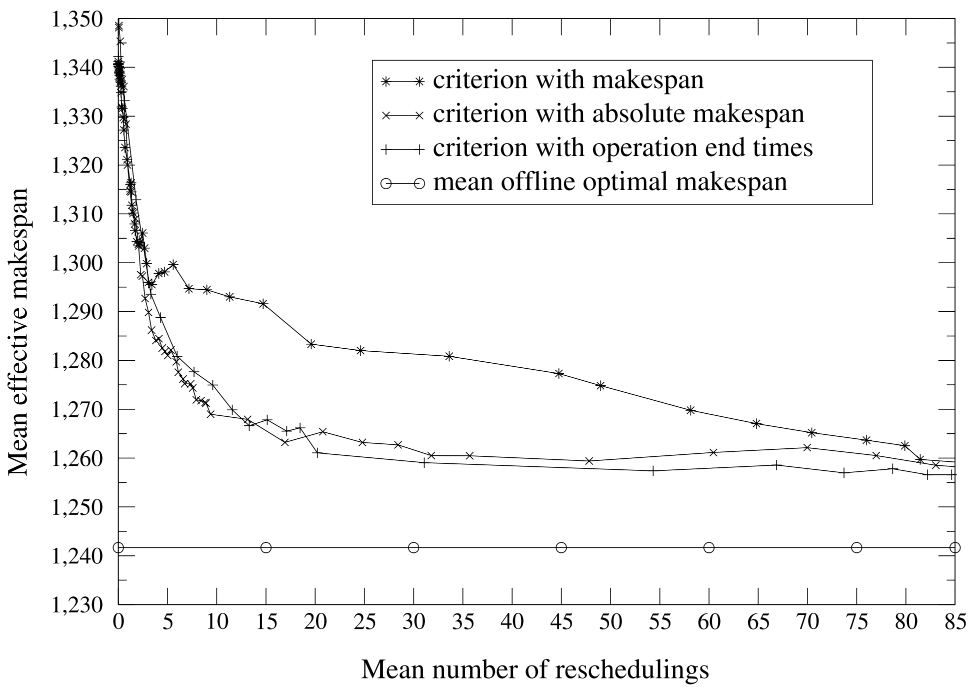


Fig. 10 Mean effective makespan for la11 with a low uncertainty

the problem la11 that consists of 100 activities (Lawrence 1984). We observe that monitoring schedule execution with the use of each rescheduling criterion improves schedule quality, i.e., mean effective makespan is smaller than the mean effective makespan obtained without rescheduling that equals 1,350: for example, after 2 reschedulings the mean effective makespan approximately equals 1,290 (4.4% of improvement). The most significant improvement is obtained with less than about ten reschedulings. The mean offline optimal makespan equals the average over the 100 optimal makespans, each of them is computed offline assuming we know the execution scenario in advance, i.e., we have to solve 100 deterministic JSSPs to optimality. Each of these 100 optimal makespans is a lower bound on the best schedule quality that can be achieved. We actually experimented with other literature JSSPs of the same size: la12, la13, la14, abz5, abz6, orb1, orb2, orb3, and orb4. The results obtained with these instances corroborate these observations (Lawrence 1984; Applegate and Cook 1991; Adams et al. 1988). These curves confirm the criterion based on the activity end times provides the smallest makespan for a given computational effort since it is more opportunistic than the first two revision criteria. We also think this is due to the fact that the activities that do not belong to the critical path have a smaller impact on makespan than the activities of the critical path because of activity slack times.

Impact of uncertainty level on makespan All experiments have been run with the same revision criterion. We chose the revision criterion revCrit3 presented above. This revision criterion performs best when about ten reschedulings are done.

Our study consists in varying the uncertainty level α and observing how the effective makespan changes. α directly changes the standard deviations of activity durations: the higher α, the larger the standard deviations. We considered the same 10 JSSP instances as those used in the study of the impact of the revision criterion on makespan. We conducted a series of three experiments, each of them corresponds to a fixed value of α. The curves represented on the next four figures have been obtained by testing 30 problem instances, 100 different realizations per problem instance. The search procedure is the same as the one presented above. The tree search is limited to one second when rescheduling and we run 1,000 simulation runs each time an activity ends.

The curves on the next three figures represent the relationship between the mean number of reschedulings and the mean relative error of effective makespan for thirty problem instances. We tested ten problem instances per uncertainty level. The mean relative error of makespan is computed over the $nbExecSce = 100$ execution scenarios of each problem instance as follows:

$$\text{mean relative error} = \frac{1}{nbExecSce} \times \sum_{sce=1}^{nbExecSce} \frac{M_{sce}^{\text{eff}} - M_{sce}^{\text{opt}}}{M_{sce}^{\text{opt}}},$$

where M_{sce}^{eff} is the effective makespan obtained after scheduling and rescheduling execution scenario sce, M_{sce}^{opt} is the optimal makespan of sce computed offline knowing sce in advance. Each M_{sce}^{opt} has been computed with a texture-based search (Beck 1999). Note that we have actually experimented with 3,000 execution scenarios since there are three uncertainty levels, 10 problem instances per uncertainty level, and 100 execution scenarios per problem instance.

Figure 11 represents the results obtained when $\alpha = 0.3$.
Figure 12 represents the results obtained when $\alpha = 0.5$.

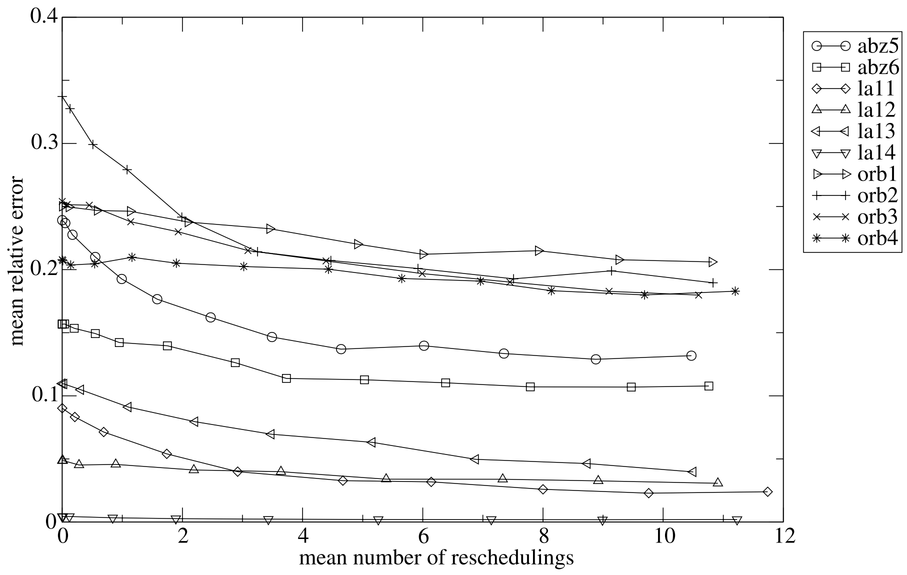


Fig. 11 Mean relative error with a low uncertainty

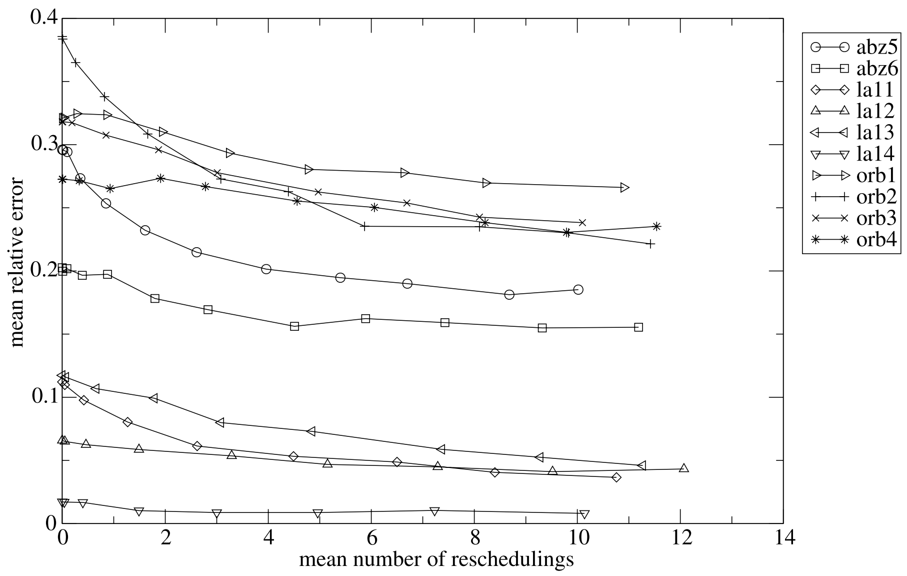


Fig. 12 Mean relative error with a medium uncertainty

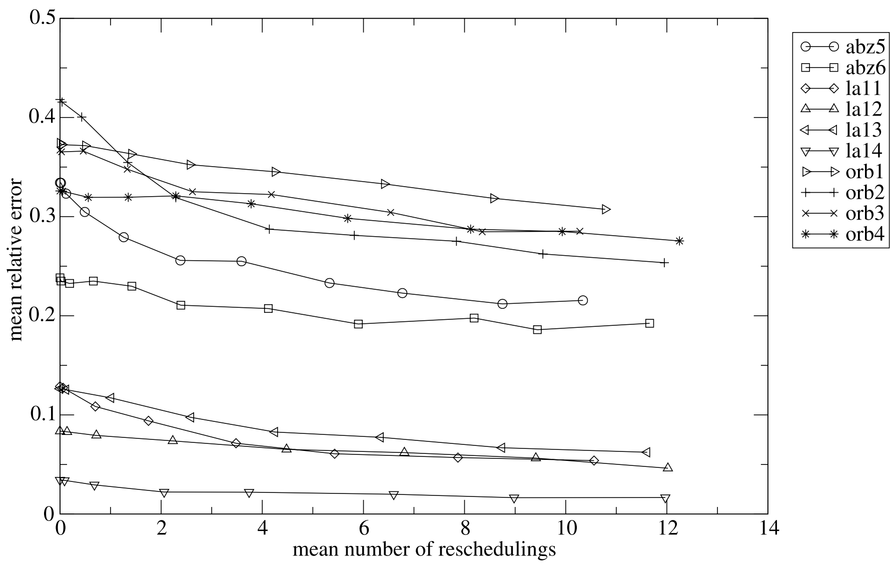


Fig. 13 Mean relative error with a high uncertainty

Figure 13 represents the results obtained when $\alpha = 0.8$.

Note that all these uncertainty levels are actually rather high since even when $\alpha = 0.3$ the standard deviation equals about the third of the mean value of the probability distribution.

Figure 14 represents the aggregated results obtained when α varies. These curves have been obtained by using linear approximation of the preceding results shown on Figs. 11, 12, and 13, and by computing the average curve over the 10 problem instances for a given uncertainty level. We have to approximate linearly the curves to perform mathematical operations on them, such as computing the average curve, because we do not control directly the number of reschedulings that are performed but we control it via sensitivity factor ω.

5.3.2 Conclusions

From the experimental results presented in Sect. 5.3.1, we conclude that it is worth using a revision approach when dealing with a JSSP with probabilistic activity durations even with a high imprecision. The higher the number of reschedulings, the lower the effective makespan. We observe that the higher the imprecision, the higher the effective makespan, and the improvement of the mean effective makespan is the same for the three uncertainty levels; moreover, it is about 6% smaller than the mean effective makespan obtained without rescheduling. Note also that the effective makespans of the schedules obtained without rescheduling are not longer than 25% of their corresponding optimal makespans on average. After about 10 reschedulings effective makespans are not longer than 18% of their

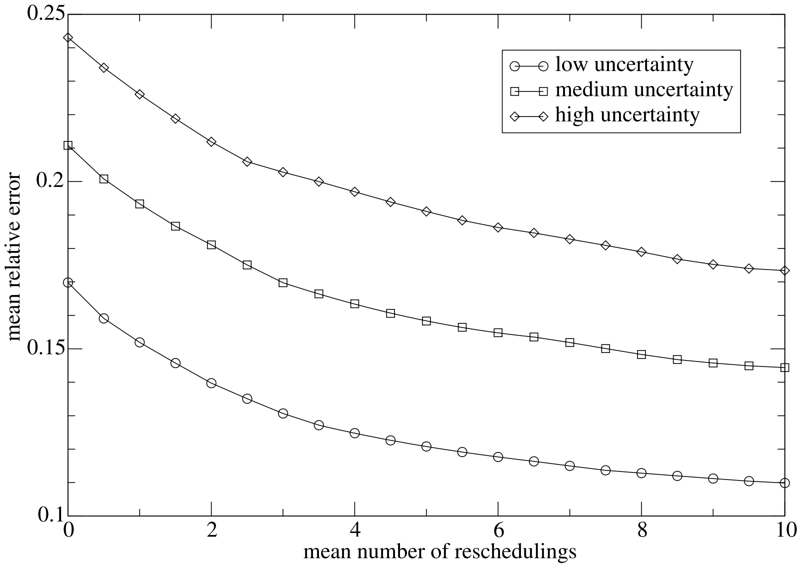


Fig. 14 Mean relative error for different uncertainty levels

corresponding optimal makespans on average. These optimal makespans are strict lower bounds since they are obtained when we know the execution scenario in advance. These lower bounds are idealistic and are, therefore, relatively far from the qualities of the best ever possible solutions to actually compute and execute.

5.4 Progressive approach

Our progressive approach is characterized by four parameters that can be set to choose indirectly the anticipation time horizon and the size of each sub-problem (commitment time horizon): $\delta t^{\min}$ controls the anticipation time horizon with respect to time, $\sigma t^{\min}$ controls the anticipation time horizon with respect to the uncertainty level, $\delta t^{\max}$ controls the size of each sub-problem with respect to time, and $\sigma t^{\max}$ controls the size of each sub-problem with respect to the uncertainty level (Bidot et al. 2006).

The problem with such an approach is that we must be very careful when taking an allocation decision or an ordering decision. (i) We have to wait until the uncertainty level around the decision is low enough so that the decision is well informed, and (ii) we cannot wait too long because we do not just want to have a reactive and myopic decision process. Determining when to select, allocate, and order a new subset of activities will be done based on monitoring a progression criterion during execution. Determining what activities to select will be done by using heuristics and Monte Carlo simulation. Determining how to allocate and order the activities of the selected subset will be done using constraint-based search, combined possibly with Monte Carlo simulation.

More precisely, suppose that we are at a given time t and we are executing a flexible schedule. We assume that a subset of activities $A_{\text{allocOrder}}$ of the problem have already been allocated and ordered given all constraints at t and the rest of the activities A_{pending} of the problem are not yet allocated and only ordered given the precedence constraints of the original problem. A subset of activities $A_{\text{executed}} \subseteq A_{\text{allocOrder}}$ have already been executed, a second one $A_{\text{executing}} \subseteq A_{\text{allocOrder}}$ are executing, and a third one $A_{\text{toBeExe}} \subseteq A_{\text{allocOrder}}$ have to be scheduled and executed. An activity is scheduled when its start time is fixed at a date before, at, or after the current time. (i) Of course, we do not want to wait until the last activity of A_{toBeExe} finishes execution before allocating and ordering subsequent activities of A_{pending} because we would then have very little time to react and could not easily come up with a good decision, and (ii) we do not want to make decisions too far in the future in regions where there is still a lot of uncertainty. Furthermore, we do not want to take the allocation and ordering decisions one by one, which would be very myopic and certainly lead to a poor schedule quality. We want to select a subset of activities and perform a combinatorial search on this sub-problem in order to satisfy temporal, resource, and cost constraints. So there are three questions here: (1) how to design conditions that will be monitored during execution and say "now, we can try to extend the current flexible schedule by allocating and ordering a new part of the problem," (2) when these conditions are met, how to select the subset of activities to be allocated and ordered, and (3) when the subset of activities is selected, how to allocate and order the activities of this subset in such a way that we maximize the probability that the constraints will be satisfied and the global cost will be minimal at the end of execution.

5.4.1 When to try extending the current flexible schedule?

To extend the current flexible schedule, we need to assess what time we still have before being forced to select, allocate, and order a new subset of activities of A_{pending} and how high the uncertainty level of the end times of the activities of A_{toBeExe} is. We need to monitor two conditions during schedule execution and when at least one of them is met, we can try to extend the current flexible schedule.

We propose to evaluate the trade-off between the fact that $\delta t^{\min}$, controlling the anticipation time horizon, should be large enough to have time to perform a combinatorial search leading to a good solution and to ensure the stability of the schedule, and the fact that execution has advanced far enough to get reduced uncertainty on the expected end times of the activities of A_{toBeExe}.

5.4.2 Temporal condition for starting selection

Given A_{toBeExe}, there exists a time point t_{D} that is the last time point before which we have to make at least an allocation decision if we want to anticipate execution. t_{D} is equal to the earliest of the expected end times of the activities of $A_{\text{allocOrderLast}} \subseteq A_{\text{toBeExe}}$ that are ordered at last positions in jobs: $t_{\text{D}} = \min_{\forall job_i \in JOB}(\max_{\forall a_{ij} \in A_{\text{toBeExe}}}(end_{ij}^{\text{exp}}))$, where the min is over all jobs job_i, the max is over all activities a_{ij} of fixed job job_i that still have to be scheduled and executed, end_{ij}^{exp} is the expected end time of activity a_{ij}, and JOB is the set of jobs. t_{D} is maintained using Monte Carlo simulation. We can try to extend the current flexible schedule from the date at which $t_{\text{D}} - t \le \delta t^{\min}$. $\delta t^{\min}$ is a parameter of the software prototype, depending on the application domain of interest.

Figure 15 represents the execution of a flexible schedule; only three jobs are partially represented; $A_{\text{allocOrder}}$ is the subset of activities represented by nine shaded rectangles; t_{D} is the earliest expected end time of the activity of job_1 ordered at the third position; $A_{\text{allocOrderLast}}$ is composed of the three activities represented by the most shaded rectangles and ordered at the last positions of jobs. A_{pending} is composed of the activities represented by white rectangles. Note that all pending activities are not represented. Precedence constraints are represented by arrows. The start times and end times of the executed activities and the start times of the executing activities are highlighted.

Uncertainty condition for starting selection When the highest standard deviation of the end times of the activities to be executed is less than a given threshold $\sigma t^{\min}$, we can select a subset of pending activities. These standard deviations are maintained using Monte Carlo simulation. In a more formal way, we extend the current flexible schedule from the date at which

$$\min_{\forall job_i \in JOB}\Big(\max_{\forall a_{ij} \in A_{\text{toBeExe}}}\big(\sigma\big(end_{ij}^{\text{exp}}\big)\big)\Big) \le \sigma t^{\min},$$

where $\sigma(end_{ij}^{\text{exp}})$ is the standard deviation of the end time of activity a_{ij}. $\sigma t^{\min}$ is a parameter of our experimental system and depends on the application domain.

5.4.3 How to select the subset of activities to be allocated and ordered?

As soon as one of the two conditions defined above is satisfied, we still need to select a subset of activities to allocate and order. We do not want to select a too large problem because: (i) we do not have an infinite time to allocate and order it (in any case less than $t_{\text{D}} - t$) and (ii) we do not want to take allocation and ordering decisions too far in the future as they concern data with too much uncertainty.

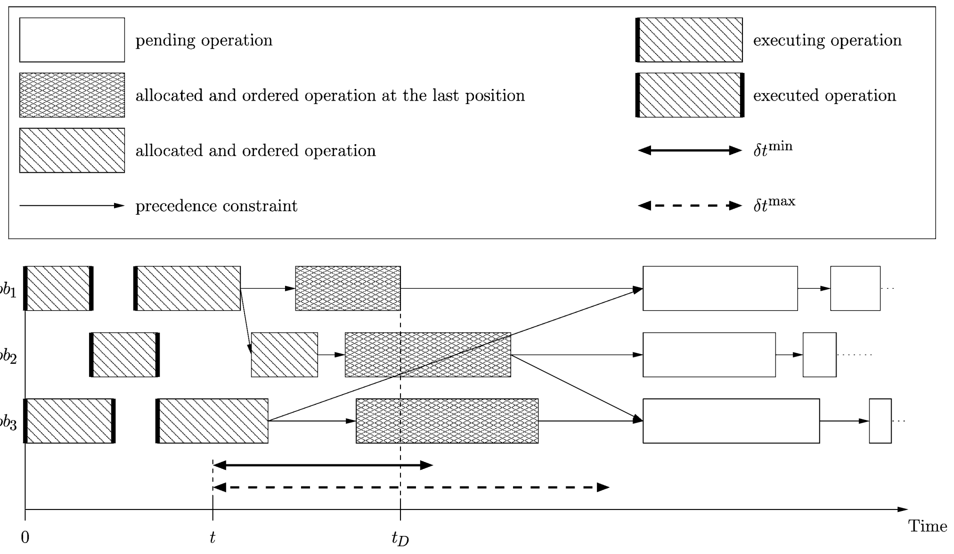


Fig. 15 An example of a progressive scheduling approach

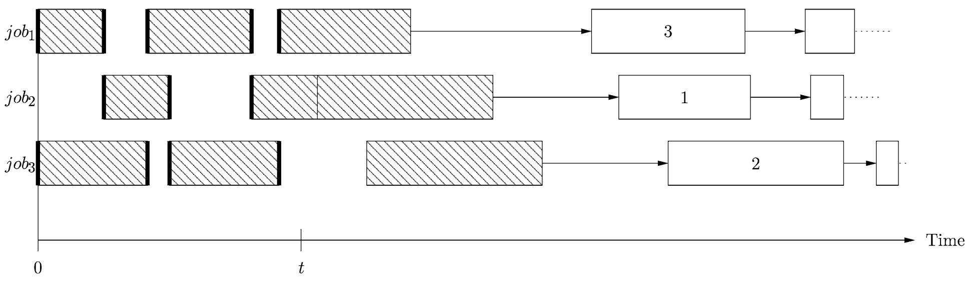


Fig. 16 Example of the assessment order of eligible activities

We thus need to monitor two conditions during the selection process. To select the subset of pending activities to be allocated and ordered, we proceed in two steps as follows: (i) we compute and associate priorities to a subset of pending activities called the eligible activities to determine the order in which we assess each of them and (ii) we assess the eligible activities by using a temporal condition and an uncertainty condition to determine which of them are selected.

Assessment order It is important to assess the eligible activities in a relevant order because we need to consider the eligible activities that have priority in terms of resource contention and tardiness costs. An eligible activity is the pending activity that is ordered at the first position of a job. There is thus one and only one eligible activity per job at the beginning of the selection process. We proceed in several steps. (1) We use a heuristic that gives the order in which we iterate through the current eligible activities and assess the current flexible schedule to determine which of them we select. (2) Once all eligible activities have been labeled by a priority value we consider each eligible activity in the decreasing order of priority and assess the probability distribution of the end time of its preceding activity with respect to the job it belongs to:

- If this distribution does not meet at least one of the two conditions defined in the next paragraph, then the eligible activity is selected and no longer eligible (and no longer pending); this selected activity is ordered and allocated in such a way that its mean end time is minimized; the next activity of the same job, which is a pending activity, becomes eligible and the priorities of the current eligible activities are then (re)computed.
- If this distribution meets the temporal condition and the uncertainty condition defined in the next paragraph, then the eligible activity remains pending and is no longer eligible concerning the current selection process. Both conditions must be met to stop the selection because we want to be sure to avoid selecting too small subsets of pending activities.

(3) If there are no more eligible activities, then the selection process is stopped, otherwise the selection process goes on by executing alternately (2) and (3).

Figure 16 gives an example of such an order; the three eligible activities are labeled in the assessment order. We compute the priority of each eligible activity by using a heuristic that is based on both an energy-based heuristic and the Apparent Tardiness Cost rule described below.

We do not need to consider the online continued execution, and we can behave as if schedule execution was stopped during the selection process since the dynamics of the underlying physical system are much slower than the dynamics of the decision-making system.

In fact, we are only interested in the probability distributions of the end times of the activities of $A_{\text{allocOrderLast}}$ and in the probability distributions of resource breakdown end times at time t when selecting and making decisions on a new subset of activities. When an eligible activity is selected, we run a set of simulations concerning this activity, its preceding activities, and the alternative resources it requires to decide what resource to allocate during the selection process. These allocation decisions are not definitive since a combinatorial search is done when the selection is finished. Figure 17 shows one simulation run of an eligible activity a_{46} associated with two alternative resources r_3 and r_4. This simulation is run at time $t = 10$. The duration of a_{46} that is randomly picked given its probability distribution equals 15 time units. The end time of its direct predecessor a_{45} is randomly picked using the current distribution and equals 25. a_{46} cannot start execution before 25 as there is a precedence constraint between a_{45} and a_{46}, and a_{46} cannot finish before 40 (this value equals the sum of its start time and its duration). Each of its alternative resources may be already allocated, not available and/or broken down. After picking activity end times, resource breakdown start times, and resource breakdown end times randomly, we can compute the expected end time of a_{46} for this simulation run. r_3 is allocated to a_{14} that finishes at 30, and r_3 breaks down at 50 and is fixed at 65. r_4 breaks down at 30 and is repaired at 40. From this simulation run, we can update the means and standard deviations of the two end times of a_{46} (a mean and a standard deviation for each resource) as follows: if a_{46} is executed with r_3, then its execution starts at 30 and finishes at 45 without interruption; if a_{46} is executed with r_4, then its execution starts at 25, it is suspended at 30, it resumes execution at 40 and finishes thus at 50. After updating

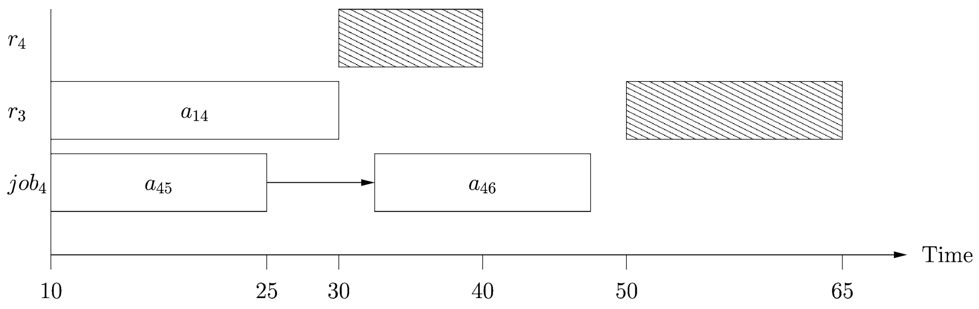


Fig. 17 Example of simulation of a selected activity

the two expected end times of a_{46} we allocate the resource that minimizes the expected end time of a_{46} (a_{46} would be allocated to r_3, if only considering this unique run).

Temporal and uncertainty conditions for stopping selection Given the assessment order, we have to make sure both that we will select enough pending activities to schedule these activities long before executing them and that the uncertainty level of the end time of each selected activity is higher than a given threshold. An eligible activity a_{ij} is selected, if the mean end time of its preceding activity of the same job a_{ij-1} is less than $t + \delta t^{\max}$ and/or the standard deviation of a_{ij-1} is less than $\sigma t^{\max}$, otherwise it is not selected and is no longer eligible during the current selection process. $\delta t^{\max}$ and $\sigma t^{\max}$ are parameters. We assume $\delta t^{\max} \geq \delta t^{\min}$ and $\sigma t^{\max} \geq \sigma t^{\min}$, otherwise we could not select any pending activity.

Energy-based heuristic The energy-based heuristic is combined with the Apparent Tardiness Cost rule based heuristic (Aktürk and Yildirim 1999) to compute the priorities of eligible activities, and the estimations of the job queues and the allocation costs of pending activities. We take resource constraints into account by extending the mean durations of the pending activities using an energy-based heuristic (Erschler 1976). We compute the criticality of each activity that depends on the average loads and costs of the resources it requires, we then modify its mean duration accordingly. We first compute the artificial duration $durRes_{ijl}$ of activity a_{ij} executed with resource r_l as follows:

$$durRes_{ijl} = \frac{mDur_{ij}}{allocCost_{ijl} \times \sum_{\forall r_k \in R_{ij}} \frac{1}{allocCost_{ijk}}},$$

where $mDur_{ij}$ is the mean duration of a_{ij}, $allocCost_{ijl}$ is the allocation cost when r_l is allocated to a_{ij}, and R_{ij} is the set of all alternative resources required by a_{ij}. Note that $durRes_{ijl}$ can be computed once offline.

It is also useful to compute the artificial duration of the pending activities requiring r_l:

$$durRes_l = \sum_{\forall a_{ij} \in A_{\text{pending}} \cap A_l} durRes_{ijl},$$

where A_l is the set of activities that require r_l. We can then compute the criticality degree $crit_{ij}$ of each pending activity a_{ij}:

$$\forall a_{ij} \in A_{\text{pending}}: \quad crit_{ij} = 1 + \sum_{\forall r_l \in R_{ij}} 1 - \frac{durRes_{ijl}}{durRes_l}.$$

$crit_{ij}$ represents how high the probability is that the effective duration of a_{ij} will be greater than the original mean duration of a_{ij} given resource constraints and allocation costs. We then extend the mean duration $mDur_{ij}$ of each pending activity $a_{ij} \in A_{\text{pending}}$ with respect to how critical a_{ij} is:

$$extendedMDur_{ij} = mDur_{ij} \times crit_{ij}.$$

Heuristic based on apparent tardiness cost rule We compute the priority of each eligible activity, the estimated job queues and the allocation costs of pending activities by using a modified version of the Apparent Tardiness Cost rule in which its weight is redefined (Aktürk and Yildirim 1999). A weight $weight_{ij}$ is associated with each pending activity. It equals the sum of the expected tardiness cost of job_i and the expected allocation cost of the eligible activity a_{ij}:

$$\forall a_{ij} \in A_{\text{eligible}}: \quad weight_{ij} = tardiCost_i^{\text{exp}} + allocCost_{ij}^{\text{exp}}.$$

The expected tardiness cost of job_i is computed as follows:

$$tardiCost_i^{\text{exp}} = \begin{cases} \phi_i \times endJob_i^{\text{exp}} - due_i, \\ \quad \text{if } endJob_i^{\text{exp}} - due_i \geq 0, \\ (1 - \phi_i) \times (endJob_i^{\text{exp}} - due_i), \\ \quad \text{otherwise,} \end{cases}$$

where $endJob_i^{\text{exp}} = end_i^{\text{allocOrderLast}} + queue_i$, and $end_i^{\text{allocOrderLast}}$ is the expected end time of the selected activity of job_i ordered at the last position. $end_i^{\text{allocOrderLast}}$ is computed by running Monte Carlo simulations.

$allocCost_{ij}^{\text{exp}}$ is the mean allocation cost associated to a_{ij}:

$$allocCost_{ij}^{\text{exp}} = \frac{\sum_{\forall r_l \in R_{ij}} allocCost_{ijl}}{k},$$

where k is the number of alternative resources required by a_{ij}. In case no new activity is yet selected for job_i,

$end_i^{\text{selectedLast}}$ is the expected end time of the allocated activity of job_i ordered at the last position and is computed by running Monte Carlo simulations. $\phi_i \in [0, 1[$ is used to balance the expected tardiness cost among the jobs. Note that $allocCost_{ij}^{\text{exp}}$ can be computed once offline. We set the look-ahead parameter $\chi = 2$, as suggested by Morton and Pentico (1993), in the formula to compute each priority π_{ij}:

$$\forall a_{ij} \in A_{\text{eligible}}: \quad \pi_{ij} = \frac{weight_{ij}}{queue_i} \times e^{\frac{-\max(0, due_i - t - queue_i)}{\chi \times \overline{queue_i}}},$$

where t is the current time, $queue_i$ is the mean duration of the pending part of job_i, and $\overline{queue_i}$ is the average over the mean durations of the pending activities of all the jobs but job_i. $queue_i$ and $\overline{queue_i}$ are computed by using the extended mean durations. In a more formal way:

$$queue_i = \sum_{\forall a_{ij} \in A_{\text{pending}} \cap A_i} extendedMDur_{ij}$$

and

$$\overline{queue_i} = \frac{\sum_{\forall job_q \in JOB \setminus \{job_i\}} queue_q}{n-1},$$

where n is the number of jobs.

5.4.4 *How to allocate and order the subset of activities?*

The set of all the selected activities on all the jobs constitutes the sub-problem to solve. After the selection process is finished, we need to approximate the contribution of each job in terms of cost, i.e., the allocation cost and the length of the pending activities of each job. This approximation is done by using the same heuristic as the one dedicated to the selection of activities. To make decisions, we generate a deterministic problem, i.e., we use the mean durations of the selected activities extended depending on resource breakdown distributions, and job queues estimated heuristically. We use standard constraint programming techniques to explore and reduce the search space.

5.5 Discussion

5.5.1 *Revision and progressive scheduling*

With respect to the experimental study of our revision approach, we set the value of the sensitivity factor that indirectly determines the number of reschedulings. When rescheduling, we are looking for solutions that optimize a criterion, there always exists a solution in our case. It could be interesting to tackle more constrained problems with the revision approach.

For the progressive approach, an experimental study has still to be conducted to understand the relationships between the different and numerous parameters, indicators, and problem characteristics.

If $\delta t^{\min}$ equals zero, then it amounts to only deciding to extend the current flexible schedule when the uncertainty condition is met. The temporal condition is met when at least one of the activities of $A_{\text{allocOrderLast}}$ finishes execution, and the uncertainty condition should be met before this date in principle, if $\sigma t^{\min}$ is not too small, see below. If $\delta t^{\max}$ and $\sigma t^{\max}$ are small, then it means we frequently extend the current flexible schedule with small subsets of activities: this is a reactive approach. If $\delta t^{\max}$ and/or $\sigma t^{\max}$ are very large, then it amounts to selecting and scheduling all activities: this is a predictive approach.

We can change both $\delta t^{\min}$ and $\sigma t^{\min}$ to choose when we want to consider a subset of pending activities during execution of the current flexible schedule. If $\delta t^{\min}$ is small and $\sigma t^{\min}$ is large, then it amounts to extending the current flexible schedule when the uncertainty condition is met. If $\delta t^{\min}$ is large and $\sigma t^{\min}$ is small, then it means we extend the current flexible schedule when the temporal condition is met. If both $\delta t^{\min}$ and $\sigma t^{\min}$ are small, then it means we extend the current flexible schedule at the last time: the temporal anticipation is short. If both $\delta t^{\min}$ and $\sigma t^{\min}$ are large, then it amounts to extending the current flexible schedule very early: the temporal anticipation is long.

5.5.2 *Integrating proactive scheduling*

We focused on revision and progressive scheduling approaches in our experimental prototypes because they are more complex than pure proactive approaches. Because revision and progressive approaches combine reasoning with execution, it is necessary to build a framework such as the one suggested here that combines generation and execution algorithms as well as the ability to sense the environment. In contrast, proactive techniques can be purely generative, building schedules to minimize some measure of expected performance and not making any assumptions about how the schedule will be executed.

In this section, we discuss a particular example of proactive scheduling from the literature, show how it can be conceptualized as an instance of our framework, and furthermore demonstrate how this conceptualization opens up a number of areas for future work which blur the boundaries between proactive, progressive, and revision approaches. As our thesis in this paper is that a system for scheduling under uncertainty needs to combine these different approaches, we see this blurring as a positive step toward an integrated scheduling system.

Proactive scheduling with uncertain durations Beck and Wilson (2007) address the job-shop scheduling problem where all activities and resource capacities are fixed but where the duration of each activity is represented as a normally distributed random variable with a known mean and standard deviation. The objective is to find a sequence of activities on each resource, consistent with the temporal constraints, that minimizes the makespan that is achievable with a given probability threshold. For example, the activity sequence that results in makespan, D, that is achievable with 0.95 probability means that in 95% of the execution scenarios for this schedule, the real makespan will be less than or equal to D. The authors seek the activity sequence that leads to the minimal such D. Unfortunately, just evaluating the probability of achieving a given makespan for a potential solution is in #P (Hagstrom 1988).

A number of algorithms are investigated, all using Monte Carlo simulation to evaluate full and partial activities sequences. A branch–and–bound method evaluates partial sequences at each node in the search tree while a set of filtering algorithms generate candidate solutions based on a deterministic approximation and evaluate them with Monte Carlo simulation.

It should be noted that the use of Monte Carlo simulation in this context is different from that discussed in Sect. 5.2. Here the simulation is used as a low complexity way of evaluating a solution to a hard problem. In Sect. 5.2, the simulation was used as part of the experimental apparatus to take the place of the "real" execution of the schedule.

Using the framework to model and extend Beck and Wilson The algorithms studied by Beck and Wilson are generation algorithms that create flexible schedules: the order of the activities on each resource is defined but no activity start times are assigned.

While, in general, a proactive technique does not require an execution context or execution algorithm, our framework does provide an interesting perspective on the algorithms proposed by Beck and Wilson. In order to evaluate a solution using Monte Carlo simulation, multiple scenarios are generated which assign a duration to each activity. Beck and Wilson evaluate the makespan of an activity sequence in a single scenario by assuming that each activity starts as soon as possible after its predecessors have ended: the start time of an activity is the maximum over the end times of all its direct predecessors.[7] Another way of viewing this assumption is as the right shift execution algorithm: each activity is shifted to the right just far enough to incorporate the realized durations of its preceding activities. In other words, a particular execution algorithm has been incorporated into the Monte Carlo simulation which evaluates the proactive schedules.

[7]Recall that the direct predecessors of an activity consist of the activity immediately preceding it in the same job and the activity immediately preceding it on the same resource.

This conceptualization immediately suggests that other execution algorithms could be incorporated. For example, rather than completely sequencing the activities on each resource, the generation algorithm might leave some activities unsequenced and rely on a more sophisticated simulation of an execution algorithm. The execution algorithm may follow the shortest expected processing time rule to decide on the complete sequence of activities and the right shift rule to set activity start times.

5.5.3 Comparison of the three families of techniques

Given information about uncertainty, an instantiation of our framework can be used, offline, with simulation, to evaluate a proactive algorithm, as in Beck and Wilson, a progressive approach, such as the one described above, or a purely revision approach, by simulating an execution algorithm. In terms of offline reasoning, *when there is uncertainty information available*, the differences among these approaches become simply a matter of degree of offline decision-making. A sophisticated system may be able to decide that in some situations (e.g., high uncertainty) a revision approach will lead to better results, while in others (e.g., low uncertainty) a more proactive approach will allow better coordination with outside suppliers.

A proactive algorithm can be used online as well, e.g., when we need to change decisions (i.e., we combine revision and proactive approaches), or when we have to make new decisions (i.e., we combine progressive and proactive approaches).

6 Adapting our framework to planning

First of all, our study which is focused on a scheduling problem is of course relevant in a more general planning framework since uncertainties on resources and time, which have been the focus in this paper, usually also play the main part in stochastic planning, and precise scheduling is an important sub-problem of planning, especially when time and resources are explicitly taken into account (Smith et al. 2000), as in plan-space and/or constraint-based planning approaches.

In some sense, the capability of dealing with conditional subsets of partially-ordered activities helps bridging the gap between scheduling and planning as such choices introduce some level of (limited) reasoning about actions, suggesting for instance to represent the problem with AND/OR graphs, as it is often done in the planning community (Nilsson 1980).

But, of course, more can be done to introduce some more planning "flavor" in the global picture. Let us look at what can be done revisiting our taxonomy:

- *Revision* planning techniques can be easily incorporated, tracking causal inconsistencies (e.g., a precondition which does not hold as expected): one can first try to adapt the plan locally, resolving the conflict through adding new constraints (e.g., strictly ordering some unconstrained activities) or inserting new activities that will restore the broken causal link; in case that fails, then complete replanning can be run.
- As it has been suggested, for instance, in Chien et al. (2000), *continuous planning* can be considered to address the arrival of new goals, simply extending the current plan to satisfy the new goals: partial-order plan-space planning is usually the best choice to address such issues.
- *Flexible* and *conditional* plans are now well-known techniques for building plans, especially in temporal constraint-based planning (Morris et al. 2001; Tsamardinos et al. 2003), which is perfectly compatible with our constraint-based scheduling framework: as we have already said, our conditional schedules are actually inspired by the literature in conditional planning.
- Last, *probabilistic* planning (Kushmerick et al. 1995) is another field of intensive research which proposes techniques to look for a *generic* solution in terms of the most probable plan to execute securely, considering distributions over known uncertainties, for instance the probability of alternative effects of some activities. Adding such probabilities (or possibilities, as in Dubois et al. 1993) and computing the overall expected probability of success of the resulting plan is no more difficult than adding probabilities on resource breakdowns or activity durations.

Our framework can hence easily be extended, both theoretically, incorporating in our model new sources of uncertainty (e.g., imprecise effects of actions) and specific algorithms for switching contexts (replanning, plan adaptation), and experimentally: techniques in the field of plan-space search, and more specifically in partial-order and constraint-based temporal planners, such as IxTeT (Lemai and Ingrand 2004) or HSTS (Muscettola 1994), are highly compatible with our constraint-based scheduling prototypes.

To conclude with this section, and focusing on the IxTeT-Exec system (Lemai and Ingrand 2004), one should notice that it does actually cover our three main types of approaches: revision (plan repair and replanning are incorporated into the execution module), proactiveness (plans are flexible) and progressiveness (new goals can be integrated at any time). Therefore, IxTeT-Exec, which lies in the area of plan-space temporal planning and uses widely constraint-based techniques, is already an example of the relevance of our work in planning. But it cannot be considered as an extension of our work, for (at least) four reasons: conditional branches are not possible in IxTeT, uncertainty on resources is not tackled, the models of uncertainty are very basic (e.g., no probabilistic nor possibilistic models), and optimization is not formally addressed in that work. We strongly believe that such work and ours could definitely benefit from each other, towards the goal of a complete and expressive formal architecture for planning and scheduling under uncertainty.

7 Conclusion and future work

In this paper, we presented a general framework for planning and scheduling when the execution environment is stochastic. Our model acts as a generic conceptual representation that can integrate three general complementary families of techniques to cope with uncertainty: proactive techniques use information about uncertainty to make decisions; revision techniques change decisions when it is relevant during execution; progressive techniques solve the problem piece by piece on a gliding time horizon. We showed this model can address diverse and complex problems, from classical scheduling settings to concerns inspired by conditional planning: in particular, it is possible to handle mutually exclusive subsets of activities. In addition, we described software prototypes directly instantiated from our model and controlled by several parameters. The prototypes can address a large range of types of uncertainties modeled with probabilities.

To summarize, the main contributions of this paper are the following:

- A three-dimension classification of the systems and techniques for planning and scheduling in a stochastic environment.
- A representation model of an extended scheduling process interleaved with execution, integrating the three families of techniques from our classification.
- Software prototypes and experimental results that support our representation when problems are probabilistic.

Apart from the extension of our model to more general planning concerns, which has been discussed in last section, interesting future work consists in enhancing our experimental system, mixing the proactive, progressive, and revision techniques and in studying the relationships between the different parameters, indicators, and problem characteristics. For example, we could set a proactive approach that takes into account 60% of execution scenarios, a middle progressive time horizon, and a small number of reschedulings. Another study could consist in setting a proactive approach that takes into account 80% of execution scenarios, a large progressive time horizon, and a small number of reschedulings. This work paves the way to the development of a software toolbox gathering a large set of algorithms to manage scheduling and schedule execution in a stochastic environment. Users of that toolbox would use the general architecture designed through our model but they would be able to design their own application by selecting relevant modules and correctly tuning parameters (e.g., anticipation time horizon, sensitivity factor, etc.). Our future work is to implement such a complete toolbox and perform additional experiments to check how parameter tuning will influence stability and robustness of the solutions that the system will generate.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job-shop scheduling. *Management Science*, *34*, 391–401.

Aktürk, M. S., & Yildirim, M. B. (1999). A new dominance rule for the total weighted tardiness problem. *Production Planning and Control*, *10*(2), 138–149.

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, *3*(2), 149–156.

Baptiste, P., & Le Pape, C. (1996). Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the fifteenth workshop of the UK planning special interest group*, Liverpool, United Kingdom.

Beck, J. C. (1999). *Texture measurements as a basis for heuristic commitment techniques in constraint-directed scheduling*. Ph.D. dissertation, University of Toronto, Toronto, Canada.

Beck, J. C., & Fox, M. S. (2000). Constraint-directed techniques for scheduling with alternative activities. *Artificial Intelligence*, *121*(1–2), 211–250.

Beck, J. C., & Wilson, N. (2007). Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*, *28*, 183–232.

Bidot, J., Laborie, P., Beck, J. C., & Vidal, T. (2003). Using simulation for execution monitoring and on-line rescheduling with uncertain durations. In *Working notes of the ICAPS'03 workshop on plan execution*, Trento, Italy, June 2003.

Bidot, J., Laborie, P., Beck, J. C., & Vidal, T. (2006). Using constraint programming and simulation for execution monitoring and progressive scheduling. In *Proceedings of the twelfth IFAC symposium on information control problems in manufacturing (INCOM 2006)* (pp. 595–600). Saint-Étienne, France, May 2006.

Bidot, J., Vidal, T., Laborie, P., & Beck, J. C. (2007). A general framework for scheduling in a stochastic environment. In *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI)* (pp. 56–61). Hyderabad, India, January 2007.

Billaut, J.-C., Moukrim, A., & Sanlaville, E. (Eds.) (2007). *Flexibility and robustness in scheduling*. Control Systems, Robotics and Manufacturing. ISTE.

Branke, J., & Mattfeld, D. C. (2002). Anticipatory scheduling for dynamic job-shop problems. In G. Verfaillie (Ed.), *Working notes of the AIPS'02 workshop on on-line planning and scheduling* (pp. 3–10). Toulouse, France, April 2002.

Bresina, J. L., Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D. E., & Washington, R. (2002). Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the 18th conference on uncertainty in artificial intelligence (UAI)* (pp. 77–84). Edmonton, Alberta, Canada, August 2002.

Chien, S. A., Knight, R., Stechert, A., Sherwood, R., & Rabideau, G. (2000). Using iterative repair to improve the responsiveness of planning and scheduling. In S. A. Chien, S. Kambhampati, & C. A. Knoblock (Eds.), *Proceedings of the fifth international conference on artificial intelligence planning and scheduling (AIPS)*. Breckenridge, CO, USA, April 2000. (pp. 300–307). Menlo Park: AAAI Press.

Davenport, A. J., Gefflot, C., & Beck, J. C. (2001). Slack-based techniques for building robust schedules. In *Proceedings of the sixth European conference on planning (ECP)*. Toledo, Spain, September 2001.

Drummond, M., Bresina, J. L., & Swanson, K. (1994). Just-In-Case scheduling. In *Proceedings of the 12th national conference on artificial intelligence (AAAI)* (pp. 1098–1104). Seattle, WA, USA, July 1994.

Dubois, D., Fargier, H., & Prade, H. (1993). The use of fuzzy constraints in job-shop scheduling. In *Working notes of the IJCAI'93 workshop on knowledge-based production planning, scheduling, and control* (pp. 101–112). Chambéry, France, August 1993.

Erschler, J. (1976). *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement*. Ph.D. dissertation, Université Paul Sabatier, Toulouse, France.

Fargier, H., Lang, J., & Schiex, T. (1996). Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of the 13th national conference on artificial intelligence (AAAI)* (pp. 175–180). Portland, OR, USA, August 1996.

Gao, H. (1995). *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, Toronto, Canada.

Geffner, H. (1998). Modeling intelligent behaviour: The Markov decision process approach. In H. Coelho (Ed.), *Lecture notes in AI: Vol. 1484*. *Proceedings of iberamia 98* (pp. 1–12). New York: Springer. Invited talk.

Hagstrom, J. N. (1988). Computational complexity of PERT problems. *Networks*, *18*, 139–147.

Herroelen, W. S., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289–306.

Hildum, D. W. (1994). *Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems*. Ph.D. dissertation, Department of Computer Science, University of Massachusetts Amherst, September 1994.

ILOG S. A. (2002). *ILOG scheduler 5.3: reference manual and user's manual*.

Kushmerick, N., Hanks, S., & Weld, D. S. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, *76*(1–2), 239–286.

La, H. T. (2005). *Utilisation d'ordres partiels pour la caractérisation de solutions robustes en ordonnancement*. Ph.D. dissertation, Institut National des Sciences Appliquées de Toulouse, Toulouse, France, January 2005.

Lawrence, S. R. (1984). *Resource-constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement)*. Ph.D. dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, USA.

Lemai, S., & Ingrand, F. F. (2004). Interleaving temporal planning and execution in robotics domains. In *Proceedings of the 19th national conference on artificial intelligence (AAAI)*. San Jose, CA, USA, July 2004.

Morris, P. H., Muscettola, N., & Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th international joint conference on artificial intelligence (IJCAI)* (pp. 494–502). Seattle, WA, USA, August 2001.

Morton, T. E., & Pentico, D. W. (1993). *Heuristic scheduling systems with applications to production systems and project management*. New York: Wiley.

Muscettola, N. (1994). HSTS: Integrating planning and scheduling. In M. Zweben & M. S. Fox (Eds.), *Intelligent scheduling* (pp. 169–212). Morgan Kaufmann: San Mateo.

Muscettola, N. (2002). Computing the envelope for stepwise-constant resource allocations. In *Proceedings of the eighth international* *conference on principles and practice of constraint programming (CP)* (pp. 139–154). Cornell University, New York, USA, September 2002.

Nilsson, N. J. (1980). *Principles of artificial intelligence*. Morgan Kaufmann: San Mateo.

Nuijten, W. P. M. (1994). *Time- and resource-constrained scheduling. A constraint-satisfaction approach*. Ph.D. dissertation, Technische Universiteit Eindhoven, Eindhoven, Netherlands.

Policella, N., Oddi, A., Smith, S. F., & Cesta, A. (2004). Generating robust schedules through chaining. In *Proceedings of the tenth international conference on principles and practice of constraint programming (CP)* (pp. 496–511). Toronto, Canada, September 2004.

Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. New York: Wiley.

Sabuncuoglu, I., & Bayiz, M. (2000). Analysis of reactive scheduling problems in a job-shop environment. *European Journal of Operational Research*, *126*, 567–586.

Sadeh, N. M. (1994). Micro-opportunistic scheduling: The Micro-Boss factory scheduler. In M. Zweben & M. S. Fox (Eds.), *Intelligent scheduling* (pp. 99–135). Morgan Kaufmann: San Mateo. Chap. 4.

Sadeh, N. M., Otsuka, S., & Schnelbach, R. (1993). Predictive and reactive scheduling with the Micro-Boss production scheduling and control system. In *Working notes of the IJCAI'93 workshop on knowledge-based production planning, scheduling, and control* (pp. 293–306). Chambéry, France, August 1993.

El Sakkout, H., & Wallace, M. (2000). Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, *5*, 359–388.

Shafaei, R., & Brunn, P. (1999a). Workshop scheduling using practical (inaccurate) data. Part 1: The performance of heuristic scheduling rules in a dynamic job-shop environment using a rolling time horizon approach. *International Journal of Production Research*, *37*(17), 3913–3925.

Shafaei, R., & Brunn, P. (1999b). Workshop scheduling using practical (inaccurate) data. Part 2: An investigation of the robustness of scheduling rules in a dynamic and stochastic environment. *International Journal of Production Research*, *37*(18), 4105–4117.

Smith, S. F. (1994). OPIS: A methodology and architecture for reactive scheduling. In M. Zweben & M. S. Fox (Eds.), *Intelligent scheduling* (pp. 29–66). Morgan Kaufmann: San Mateo.

Smith, D. E., Frank, J., & Jónsson, A. K. (2000). Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, *15*(1), 61–94.

Tsamardinos, I., Vidal, T., & Pollack, M. E. (2003). CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, *8*(4), 365–388.

Vidal, T., Ghallab, M., & Alami, R. (1996). Incremental mission allocation to a large team of robots. In *Proceedings of the IEEE international conference on robotics and automation (ICRA'96)* (Vol. 2, pp. 1620–1625). Minneapolis, MN, USA.

Wang, X., & Chien, S. A. (1997). Replanning using hierarchical task network and operator-based planning. In *Proceedings of the fourth European conference on planning (ECP)* (pp. 427–439). Toulouse, France, September 1997.

Wu, S. D., Byeon, E.-S., & Storer, R. H. (1999). A graph-theoretic decomposition of the job-shop scheduling problem to achieve scheduling robustness. *Operations Research*, *47*, 113–123.