A General Framework for Scheduling in a Stochastic Environment[*]

Julien Bidot
Universität Ulm
Ulm, Germany
julien.bidot@uni-ulm.de

Thierry Vidal
ENIT
Tarbes, France
thierry@enit.fr

Philippe Laborie
ILOG S.A.
Gentilly, France
plaborie@ilog.fr

J. Christopher Beck
University of Toronto
Toronto, Canada
jcb@mie.utoronto.ca

Abstract

There are many systems and techniques that address stochastic scheduling problems, based on distinct and sometimes opposite approaches, especially in terms of how scheduling and schedule execution are combined, and if and when knowledge about the uncertainties are taken into account. In many real-life problems, it appears that all these approaches are needed and should be combined, which to our knowledge has never been done. Hence it it first desirable to define a thorough classification of the techniques and systems, exhibiting relevant features: in this paper, we propose a tree-dimension typology that distinguishes between proactive, progressive, and revision techniques. Then a theoretical representation model integrating those three distinct approaches is defined. This model serves as a general template within which parameters can be tuned to implement a system that will fit specific application needs: we briefly introduce in this paper our first experimental prototypes which validate our model.

1 Introduction

Many approaches to scheduling assume an execution environment without uncertainty. The problem there is to allocate resources and assign start times to a set of given activities, so that temporal and resource constraints are satisfied. The resulting *predictive schedule* is then sent to the execution controller. However, in practical applications, we have to schedule with incomplete, imprecise, and/or uncertain data: simply executing a strictly and completely determined predictive schedule is not a right answer anymore, as there is high chance that such a schedule will not fit the real situation that will arise. Scheduling and schedule execution can then be reconsidered in many ways: for instance, setting activity start times of a sequence of activities can be postponed until execution, adding some flexibility to a schedule that will better adapt to observed events; the execution controller can be augmented by rescheduling capabilities in case a failure occurs; or schedule generation and execution can be interleaved so as to only predict in a short range, where uncertainty remains low enough. Such approaches are very different from one another, and it is still unclear which one is the best answer in a given application context: in this paper, we will propose (in Section 3) a thorough classification of techniques for scheduling under uncertainty, exhibiting their strengths and weaknesses. Our conclusion is that in real-life applications, mixing those techniques within a single system appears to be highly desirable. For that purpose we will propose (in Section 4) a conceptual theoretical model encompassing both the generation and the execution of the schedule and in which a large number of techniques for dealing with uncertainty can be concurrently implemented. Section 5 will present experimental prototypes that partially validate our model.

2 Background

A standard scheduling problem comprises a set of activities and resources. Each activity has a duration, and each resource has a limited capacity. The objective is to assign resources and times to activities given temporal and resource constraints. In general, scheduling problems are optimization problems: typical optimization criteria are makespan, number of tardy activities, tardiness or allocation cost. If we assume an execution environment without uncertainty, we generate a *predictive* 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 e.g. resources may break down).

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

Definition 2.1 (flexibility) *A flexible schedule is a schedule that is not fully set: decisions have still to be made.*

Decisions to make or to change can be of heterogeneous types; e.g., allocation or sequencing decisions.

Definition 2.2 (conditional schedule) *A conditional schedule is a special kind of flexible schedule in which distinct alternative subsets of partially ordered activities can be modeled: the remaining decisions here are to choose between such alternatives at execution time.*

[*]Partially supported by Convention Industrielle de Formation par la REcherche 274/2001.

Definition 2.3 (executable schedule) *An executable schedule is a schedule that does not violate any constraint.*

Definition 2.4 (adaptive scheduling system) *An adaptive scheduling system is a system that is able to generate a new executable schedule whenever the current executing schedule is no longer executable.*

Definition 2.5 (robustness) *A robust schedule is a schedule whose quality (according to the optimization criterion) does not deviate too much during execution with respect to known online perturbations: the less deviation, the more robustness.*

Definition 2.6 (stability) *A stable schedule is a schedule in which no decision will be changed during execution.*

3 Classification

In this section, we concisely describe a taxonomy of scheduling systems or techniques that is independent of any specific representation model or reasoning technique. Such classifications have already been done, especially in the Operations Research community (as in Herroelen *et al.* [2004]), but none is totally satisfactory to our need since they only distinguish between offline and online techniques. We go beyond this distinction and consider more issues such as how and when decisions are made, optimality requirements, etc.

3.1 Proactive Techniques

A proactive technique takes into account uncertainty to produce schedules that are less sensitive to online perturbations.

A first naive method for making a schedule insensitive to online perturbations is to produce offline a unique predictive, robust schedule by taking into account the worst scenario.

Another approach consists in introducing some flexibility in the schedule: only a subset of decisions are made offline with a search and the rest online without 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 plan 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 an *incomplete flexible schedule*. Another way is to build an *undecided flexible schedule*: everything is set but with alternative branches, leading to a conditional schedule.

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).

3.2 Revision Techniques

Revision techniques consist in changing decisions during execution when it is necessary; e.g., we change decisions when the current predictive schedule becomes inconsistent, when estimated quality deviates too much from the predicted 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; e.g., Sadeh *et al.* [1993] developed both simple rules to adapt a current schedule through e.g. activities shifting, and more elaborated local rescheduling techniques when the problem is more acute.

3.3 Progressive Techniques

The idea behind progressive techniques is to interleave scheduling and execution, by solving the whole problem piece by piece, where each piece corresponds to a time horizon slice. 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; e.g., Vidal *et al.* [1996] 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 first available. A decision is made when uncertainty level of the used information is not too high, and/or when the *anticipation horizon*, the interval between the current time and the expected end time of the last scheduled activity, has become too small. One thus needs an execution-monitoring system able to react and indicate when, what, and what type of new decisions to make.[1]

3.4 Discussion

We can compare the three families of techniques with respect to the following criteria: online memory need, online CPU need, schedule quality/robustness, and stability.

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. We can expect a very high schedule quality if we are able to reoptimize very often and globally revise the current schedule, but robustness is not guaranteed. Stability may be very bad if we change a lot of 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 power keeps low since we do not have to search for solutions with backtracking. We can expect a high schedule robustness or stability since we take into account what may occur online to make decisions.

Progressive techniques permit to limit our online memory need to the minimum since we only store a piece of schedule. The requirement in CPU online is limited since we only solve sub-problems. It is difficult to guarantee a very high schedule quality/robustness since decisions are made without taking into account knowledge about uncertainty to make scheduling decisions and with a more or less short-term/aggregated view. This family of techniques generates stable schedules.

These features can help a decider to choose a technique in a specific application domain: if memory usage is limited,

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

then conditional schedules are probably not the right answer. Moreover, one can easily see that mixed techniques are necessary: for instance, in a highly stochastic world, the time spent on rescheduling can be reduced when a proactive or a progressive approach is used, but on the contrary a pure proactive technique is not realistic since there will always be unpredicted or unmodeled deviations that can only be dealt with by a revision technique. The combinatorial explosion of conditional schedules also suggests to develop only some of the branches and add more piece by piece in a progressive way. As a matter of conclusion, a decider should also be given a global system encompassing all of the three kinds of approaches, allowing her to tune the levels of proactivity, progression, and revision that will best fit her needs. A 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.

4 Representation Model

This section describes a generic representation model for scheduling in a stochastic execution environment. This model integrates the three families of approaches presented in the previous section.

4.1 Schedule

We are interested in extended scheduling problems with mutually exclusive subsets of activities, in a way similar to what was done in Tsamardinos *et al.* [2003]. 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 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 when a decision is made or when a decision is propagated *via* constraints.

We distinguish two types of variables in the problem: the *controllable variables* and the *contingent variables*.[2]

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

Decisions may influence the state of the environment. One of the issues that depends on application domains is to decide when to instantiate controllable variables. For example, it is difficult to set activity start times in advance when activity durations are imprecise because of temporal constraints.

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

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

Moreover, a set of (probabilistic/possibilistic/etc.) distributions of possible values may be attached to each contingent variable. Such distributions are updated during execution.

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 the same time. 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 states 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 $ay = \langle start_{ay}, d_{ay}, end_{ay}, [\mathcal{CT}_{ay}]\rangle$ is defined by three variables: a start time variable $start_{ay}$, a duration variable d_{ay}, and an end time variable end_{ay}. These variables are either controllable or contingent. ay may be associated with an optional set of resource constraints $\mathcal{CT}_{ay}$ that involve the variables of the resources it requires.*

In a constraint-based model, to propagate the bounds of the variable domains, we usually post the following constraint for each activity: $end_{ay} - start_{ay} \geq d_{ay}$. Of course, constraints of any type between variables can be posted on our scheduling problem.

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 Section 3, additional constraints may have to be posted by the schedule generation algorithm to (more or less) set resource allocations, make sequencing decisions, and set precise activity start times. Hence we do not need to add anything to our model to achieve this.

Central to our model is the notion of *conditions* that are subsets of variables related by logical and/or mathematical 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 mathematical 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*; i.e., a branching condition is a condition that will be attached to one of mutually exclusive subsets of activities (see below). Such a condition will be checked at a specific time-point that we will call a *branching node*.

We propose the following recursive definition of a schedule to describe our model with respect to these particular mutually exclusive subsets of activities.[3]

Definition 4.8 (schedule) *A schedule $\mathcal{S}$ is either*
• *void $\mathcal{S} = \emptyset$, or*
• *$\mathcal{S} = \langle ay_{\mathcal{S}}, \{\mathcal{CT}_{\mathcal{S}}\}^*, \mathcal{S}'\rangle$ is an activity $ay_{\mathcal{S}}$ 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 $cnd_{\mathcal{S}}$. A node is a dummy activity ay^{dum} of null duration that does not require any resource: $ay^{\mathrm{dum}} = \langle start_{ay}^{\mathrm{dum}}, 0, end_{ay}^{\mathrm{dum}}\rangle$, with $start_{ay}^{\mathrm{dum}} = end_{ay}^{\mathrm{dum}}$.*

Definition 4.9 (recipe) *A recipe $rcp = \langle \mathcal{S}, [Py_{rcp}], bc_{rcp}\rangle$ is 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 product can be made 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 two alternatives to define recursively a classical partially ordered schedule without alternatives. The third introduces parts of a schedule that divide, at some given time-point, 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.

The previous recursive definitions are actually constructive definitions that permit 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 is no temporal constraint between two activities that do not belong to the same recipe. However, some precedence constraints can be added to constrain branching conditions to be checked before their related recipes would be executed.

4.2 Generation and Execution

We only defined a model that a proactive method could use to generate a more or less flexible 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 now a dynamic problem in which a solution is executed in a stochastic environment, thus requiring more scheduling decisions to be made 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 flexible schedule (as defined in previous section) 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).

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. The branching condition defined in the previous section is actually used by the execution algorithms, guiding them into the adequate alternative. We need to introduce here two new types of condition: an *activation 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 activation and fire conditions are needed both in revision and progressive approaches.

Typical examples of activation 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 more simply a condition stating that the anticipation horizon becomes 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 an *automaton* whose states 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}.*

An execution context is a schedule that is a solution of the whole scheduling problem or a part of it. In addition, an execution context may not contain all recipes starting from a branching node, but only those with the highest values Py; another example of activation condition is hence that when the value Py of a remaining recipe becomes high enough, that recipe should be developed and included in the current schedule: we hence generate in a progressive way a new schedule which is the current one augmented with an additional recipe.

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

Our automaton also includes transitions for generating execution contexts and going from one execution context to another one.

Definition 4.11 (transition) *A transition $tr = \langle ect_{tr}^{src}, ect_{tr}^{tat}, cond_{tr}^{\mathrm{act}}, cond_{tr}^{\mathrm{fir}}, \beta_{tr}\rangle$ is composed of*

[3]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, a schedule implying in the Operations Research community that all start times are set.

a source execution context ect_{tr}^{src}, a target execution context ect_{tr}^{tat}, an activation condition $cond_{tr}^{\mathrm{act}}$, a fire condition $cond_{tr}^{\mathrm{fir}}$, and a generation algorithm β_{tr}.

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

Transition tr is activated when its activation condition is met. When tr is activated, 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. β_{tr} may be run offline or online; it 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}. Activation condition $cond_{tr}^{\mathrm{act}}$ must be more general than or equal to fire condition $cond_{tr}^{\mathrm{fir}}$ since $cond_{tr}^{\mathrm{act}}$ 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 be fired in an automaton model; e.g., a template transition associated with a resource constraint rct_1 may be fired each time one of the activities involved in rct_1 is executing and allocated to the resource involved in rct_1.

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 contexts. When an activation 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 find at least one schedule, if not the optimal one. 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.

As a matter of conclusion, one can see any 'pure' technique can be easily instantiated with our model: a pure proactive technique will barely need a single context, generation being made once for all offline, the remaining decisions being taken by the sole execution algorithm; in a pure revision (resp. progressive) approach, contexts contain non-flexible predictive schedules with basic execution algorithms, and activation/fire conditions associated with failures or quality deviations in the current context (resp. to the horizon getting too small or the uncertainty level decreasing), and generation algorithms change the current schedule to fit the new situation (resp. add a new subset of activities). But the great 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.

5 Experimental System

In this section, we simply recall a few software prototypes that we implemented, and we show how they are actually special cases of our global model and hence partly validate it. Experimental results appear in the cited papers.

5.1 Scheduling Problem

The *flexible job-shop scheduling problem* (flexible JSP) is a scheduling problem where the set of activities AY is partitioned into *jobs*, and with each job is associated a total ordering over a subset of AY. Each activity specifies a set of alternative resources on which it must execute without interruption. No activities that require the same resource can overlap their executions. We represent this formally by a partition of the set of AY into *resource sets*. 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 AY.

For our experimental investigations, we focused on probabilistic flexible job-shop problems. Random variables are fully independent and associated with probability distributions. We conducted experiments with two criteria to minimize: makespan, and sum of tardiness and allocation costs.

5.2 Architecture

Our experimental system is composed of the following modules: a solver, a controller, and a world.[4] The solver module is in charge of making decisions with a backtrack search, constraint propagation, and Monte-Carlo simulation. The decisions made by the solver module are sent to the controller module. The latter is responsible for choosing activity start times given decisions made by the solver module 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 is

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

also in charge of maintaining known and unknown probability distributions running Monte-Carlo simulation online.

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; e.g., 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 and using simulation for the non-executed part of the schedule, and we compare this absolute difference with a reference value. 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].

5.4 Proactive Approach

Our experimental proactive approach is set by two main parameters. The first parameter $proact_{\mathrm{gene}}$ is used to generate a problem model without uncertainty from the stochastic problem model; i.e., we choose a value for each random variable. The greater the values chosen, the more proactive the technique. A possibility is to choose and use the average values of distributions. The second parameter $proact_{\mathrm{simu}}$ is Boolean and determines whether Monte-Carlo simulation is used or not during search. Moreover, the number of simulation runs can be chosen [Beck and Wilson, 2005].

5.5 Progressive Approach

Our progressive approach is characterized by four parameters that can be set to choose indirectly the anticipation horizon and the size of each sub-problem: $\delta t^{\min}$ controls the anticipation horizon with respect to time, $\sigma t^{\min}$ controls the anticipation 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].

6 Conclusion and Future Work

In this paper, we presented a general framework for scheduling when execution environment is stochastic. Our representation model acts as a generic conceptual model that can integrate three general complementary families of techniques to cope with uncertainty: proactive techniques use information about uncertainty to generate and solve a problem model; 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 scheduling problems; in particular, it is possible to handle mutually exclusive subsets of activities. In addition, we described software prototypes directly instantiated from our representation model and controlled by several parameters. The prototypes can address a large range of probabilistic scheduling problems. 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 horizon, sensitivity factor, etc.). Our future work is to implement such a complete toolbox and make additional experiments to check how parameter tuning will influence stability and robustness of the solutions that the system will generate.

References

[Beck and Wilson, 2005] J. C. Beck and N. Wilson. Proactive algorithms for scheduling with probabilistic durations. In *Proc. of the 19th Int'l Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, July 2005.

[Bidot *et al.*, 2003] J. Bidot, P. Laborie, J. C. Beck, and T. Vidal. 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 *et al.*, 2006] J. Bidot, P. Laborie, J. C. Beck, and T. Vidal. Using constraint programming and simulation for execution monitoring and progressive scheduling. In *Proc. of the Twelfth IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2006)*, Saint-Étienne, France, May 2006.

[Fargier *et al.*, 1996] H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI'96)*, Portland, Oregon, USA, August 1996.

[Herroelen and Leus, 2004] W. S. Herroelen and R. Leus. Robust and reactive project scheduling: A review and classification of procedures. *Int'l Journal of Production Research*, 42(8):1599–1620, 2004.

[Morris *et al.*, 2001] P. H. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proc. of the 17th Int'l Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, Washington, USA, August 2001.

[Sadeh *et al.*, 1993] N. M. Sadeh, S. Otsuka, and R. Schnelbach. 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*, Chambéry, France, August 1993.

[Tsamardinos *et al.*, 2003] I. Tsamardinos, T. Vidal, and M. E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *CONSTRAINTS*, 8(4), 2003.

[Vidal *et al.*, 1996] T. Vidal, M. Ghallab, and R. Alami. Incremental mission allocation to a large team of robots. In *Proc. of the 1996 IEEE Int'l Conference on Robotics and Automation (ICRA'96)*, Minneapolis, Minnesota, USA, April 1996.