# THANK YOU
**FOR YOUR ORDER.**

D:IBM or IBM subsidiary
Philippe Laborie
rue de Verdun
Gentilly, 94253
France

IBM KnowledgeGate Operations
TB-17C, M/D T-08 One North Castle Drive
Armonk, NY 10504
United States
Phone  or
Email itorders@us.ibm.com

**ORDER INFORMATION**

| | |
|---|---|
| INFO #: | 19827503 |
| SHIP VIA: | Ariel (eMail TIFF) |
| ORDER #: | |
| CUSTOMER #: | 123283 / 2285751 |
| BILLING REF: | |
| CCD: | 4590 |
| ORDERED: | 05/02/2011 |
| NEED BY: | |

**ARTICLE INFORMATION**

| | |
|---|---|
| ACCESSION #: | 09565515 |
| PUBLICATION: | JOURNAL OF INTELLIGENT MANUFACTURING |
| DETAILS: | 21(1): 2010 |
| AUTHOR: | |
| TITLE: | Validating scheduling approaches against executional uncertainty |

**CUSTOMER INFORMATION**

| | |
|---|---|
| SPECIAL INSTRUCTIONS: | |
| COMPANY: | D:IBM or IBM subsidiary |
| PATRON: | Philippe Laborie |
| PHONE: | 0139083567 |
| FAX: | |
| EMAIL: | laborie@fr.ibm.com |
| ARIEL: | |

***The document you requested from KnowledgeGate is being sent to you via Ariel electronic delivery. Ariel sends a .tif file attachment to your email address. Opening the attachment requires a multipage TIFF viewer included with most operating systems (or download at http://www.alternatiff.com).***

***To access your document, click on the right arrow button on the tools bar. Use the right and left arrow buttons to navigate through your document.***

***Only the attachment called "Document.TIF" holds your article. The other two hold information about delivery. There is no need to open them unless you need more information about your delivery.***

***This document is protected by U.S. and International copyright laws. No additional***

# REGULAR

J Intell Manuf (2010) 21:49–64
DOI 10.1007/s10845-008-0172-7

# Validating scheduling approaches against executional uncertainty

**Riccardo Rasconi · Amedeo Cesta · Nicola Policella**

Received: 5 August 2008 / Accepted: 14 August 2008 / Published online: 18 November 2008


**Abstract** This paper introduces a general methodology to perform a comparative evaluation of different approaches to the problem of scheduling with uncertainty. Different proactive (off-line) and reactive (on-line) scheduling policies are evaluated by simulating the execution of a number of baseline schedules under uncertain environmental conditions, and observing the solution behaviors as such schedules get stressed by exogenous events. The analysis aims at assessing the impact of both proactive and reactive scheduling efforts on the robustness of the baseline solutions, against measurable disrupting factors, through reproducible experiments. As the results show, this dynamic approach reveals extremely useful to unveil some subtle aspects, which would have remained undetected through static metric evaluations.

**Keywords** Scheduling with uncertainty · Reactive scheduling · Proactive scheduling · Schedule execution

## Introduction

A promising area for scheduling methodologies is related to the automation-supportive aspects of schedule control at execution time. In this respect, a schedule is interpreted as a set of actions through which the evolution of the environment can be "guided" toward a desired status. Because of its several real-world applications, the scheduling problem has been widely studied by many scientific communities; the growing attention dedicated to this specific issue in research areas such as OR and AI is also proved by the increasing number of single results and surveys, (e.g., Mc Kay et al. 1988; Aytug et al. 2005; Herroelen and Leus 2004).

Yet, these different approaches share a common drawback: they tend to not sufficiently emphasize the need to execute the schedules in real working environments, where a variety of possible events may invalidate the current schedules making some proper and quick adjustments necessary.

In practice, it is essential that the evolution of an executing schedule complies with a determined set of conditions; during the execution of a schedule it is of great importance to maintain such conditions satisfied by triggering proper rescheduling actions in spite of possible executional disturbances (Smith 1994). This *control* exerted on the schedule has a direct influence on the evolution of the real physical system, and a correct schedule management is necessary to maintain the system's characteristics within the required behavioral bounds.

Before being executed, a baseline set of actions has to be initially scheduled; the objective of this off-line phase is the synthesis of a solution that retains a set of desired characteristics, as these features represent the warranty of good executional performance. Depending on the desired behaviors, different emphasis can be given to the off-line or to the on-line phase of the solving process.

All this considered, it is initially worth to introduce a broader definition of scheduling problem which must comprise the following two aspects:

- the *static sub-problem*. Given a set of *activities* (or tasks) and a set of *constraints*, it consists in computing a consistent assignment of start and end times for each activity. The solution of this problem is computed according

R. Rasconi (✉) · A. Cesta
Institute for Cognitive Science and Technology, Italian National Research Council, Rome, Italy
e-mail: riccardo.rasconi@istc.cnr.it

N. Policella
European Space Agency, European Space Operations Center, Darmstadt, Germany
e-mail: nicola.policella@esa.int

to some optimization criteria, that depend on the quality measures of interest.
- the *dynamic sub-problem*. It consists in monitoring the actual execution of the schedule and repairing the current solution, every time it is necessary. The need to revise the schedule arises as a consequence of *exogenous event* occurrences.

In the remainder of the paper, the term "scheduling" is used according to a broader definition of scheduling problem that involves both the off-line and the on-line phases while by "scheduling approach" we mean the pair ⟨*off-line procedure, on-line procedure*⟩.

According to the previous definition, each scheduling procedure can be placed in a bidimensional space, where the values on the $x$-axis and $y$-axis are proportional, respectively, to the extent of the on-line and off-line efforts associated to the employed solving methodologies (Fig. 1). As the schema shows, the scheduling methodology with the lowest on-line component is the *Synthesis of Robust Solutions* (Leon et al. 1994; Jensen 2001; Sevaux and Sörensen 2002): these approaches exploit the partial knowledge of the possible exogenous events to build solutions capable of absorbing the environmental uncertainty. Once the schedule is synthesized and dispatched for execution, uncertainty counteraction is mostly demanded to the solution's robustness. Obviously, no scheduling technique could lie entirely on the off-line axis, as the produced solutions would be meaningful only under the unrealistic condition of a completely deterministic execution environment.

On the contrary, the approach that produces *Partially Defined Schedules* (Wu et al. 1999; Roy and Sussman 1964) does not use any knowledge on the uncertainty. According to this methodology, the schedules are characterized by partially ordered activities, therefore maintaining the possibility to dynamically re-adjust the temporal mutual relationships among the tasks, as the unforeseen events occur. As Fig. 1 shows, this approach implies either an off-line phase and a significant on-line phase: in fact, a considerable effort must be spent on the synthesis of the partially ordered baseline schedule; thereafter, the same schedule must be properly managed through specialized techniques during the execution phase, so as to take advantage from the partial ordering of the tasks.

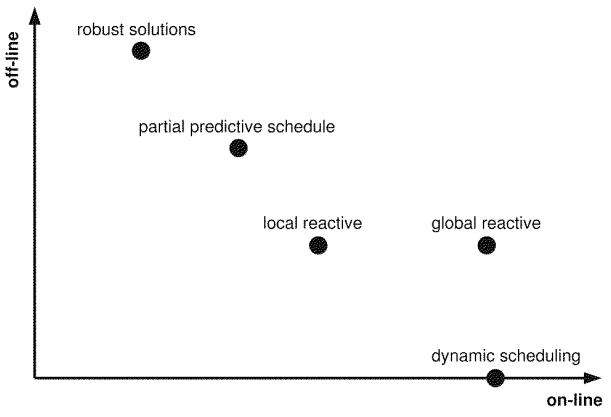


**Fig. 1** Various approaches to scheduling, classified on the basis of the off-line and on-line phases

A greater effort in the on-line phase is required in the *Reactive Scheduling* approaches. According to this strategy, a schedule is supposed to be *rescheduled* during its execution, whenever the occurrence of exogenous events might spoil its consistency. The goal of the rescheduling action is therefore to preserve (a) the set of initially imposed temporal relations among the problem activities, and (b) the general temporal and resource consistency despite the varying environmental conditions. Depending on the number of activities involved in the rescheduling procedure, the reactive scheduling strategy is defined *local* or *global*. Local reaction (Smith 1994) involves the rescheduling of a limited number of tasks, while global reaction (El Sakkout and Wallace 2000) may generally span over the whole set of the schedule activities: obviously, the two techniques entail a different level of on-line effort and consequently generate solutions of different quality.

*Dynamic scheduling* is remarkable as it has no off-line component, and all scheduling decisions are taken in the on-line phase. No baseline schedule is initially computed, and all the activities of the plan are dynamically dispatched for execution as conditions allow.

*The problem* The problem we intend to tackle in the present work is to analyze the effectiveness of different scheduling approaches in solving reactive scheduling problem instances. The issue at stake here is not only related to finding a suitable initial solution, but also how to properly manage the solution as it gets executed in an environment where unexpected events may occur and degrade the baseline schedule's original characteristics to the point of possibly spoiling its consistency.

Within this contextual frame, analyzing the execution of a schedule entails a search on a four-dimensional space, where each dimension represents the domain of a potentially independent variable. In particular, the problem variables are: (1) the solution's feature of interest $\mu$ (makespan, reactivity, continuity, etc.), (2) the degree of proactiveness of the employed strategy (y-axis, Fig. 1), (3) the degree of reactiveness of the employed strategy (x-axis, Fig. 1), and (4) the impact factor of real-world uncertainty (i.e., the *difficulty* of the reactive scheduling problem instance it represents).

The analysis of any scheduling approach characterized by an on-line component obviously entails a dynamic testing within simulated execution conditions. *The key idea is to define the reactive scheduling problem in terms of the disturbances that can be injected to a given base scheduling*

*problem instance during its execution*. Each disturbance adds up to represent environmental uncertainty, and will act to modify the base problem's initial structure within some extent: as we will show, the difficulty of each reactive scheduling problem instance can therefore be assessed in terms of how much the base problem's initial characteristics deviate from the initial values; the quality of the rescheduling procedures can be assessed depending on how well they succeed in restoring/maintaining such desired characteristics in the solution as the outcome of the revision process.

*Paper outline* The analysis requires a clear acknowledgment of the following main points:

(1) how can we assess and/or compare the efficacy of different scheduling strategies?
(2) How can we quantify the stress-factor of one or more executions? In other words, how can we objectively assess the difficulty of a reactive scheduling problem?

Before answering these two questions, we provide some necessary definitions in section "The reactive scheduling problem"; subsequently, the first of the two previous issues is discussed in section "Measuring the scheduling efficacy", while the problem of synthesizing reactive scheduling problem benchmark instances of controllable difficulty (Policella and Rasconi 2005) is tackled in section "Generating benchmarks for the reactive problem". The two previous points are strictly related: obviously, a true comparison of the efficacy of two on-line scheduling actions must be performed on the basis of reactive scheduling problems of equal difficulty. As will be shown, both aspects are related to the introduction of a suitable measure of "Schedule Robustness".

Section "Simulating uncertainty during execution" is dedicated to the problem of modeling environmental uncertainty through the introduction of exogenous events templates; in section "A framework for schedule dynamic analysis" we identify a set of different off-line and on-line strategies and introduce a general schedule execution management system where they can be analyzed, while section "Designing an experimental analysis" finally presents an overview of the empirical analysis that has been performed within the execution management system, as well as an explanation of the obtained results.

## The reactive scheduling problem

In this section we provide a formal definition of the Reactive Scheduling Problem, in terms of the Project Scheduling Problem; the same definitions will be subsequently used in this document, where we provide an analysis of the dynamic sub-problem by (a) selecting a number of particularly meaningful events which are likely to occur during schedule execution, and (b) showing how these events may represent the building blocks for the synthesis of reactive scheduling problem benchmarks.

In this perspective, the reactive scheduling problem is most naturally represented as an optimization problem. In this analysis, we focus on a particular family of scheduling problems, known as *Project Scheduling* problems, which may be defined as follows:

**Definition 1** (*Project scheduling problem—PSP*) The Project Scheduling Problem can be formalized as a tuple $\langle \mathcal{V}, \mathcal{C}, \mathcal{R} \rangle$ where:

- $\mathcal{V}$ is the set of activities that have to be scheduled. Each activity $a_i \in \mathcal{V}$ is characterized by a start time $s_i$ and a duration $p_i$;
- $\mathcal{C}$ is the set of temporal constraints that exist between activity pairs $\langle a_i, a_j \rangle$. The temporal constraints impose limitations on either single and mutual allocations in time of the activities to be scheduled;
- $\mathcal{R}$ is the set of renewable resources, each characterized by a maximum capacity $C_k^{max}$; the fact that the resources have limited capacity, represents a further constraining factor for the temporal allocations of the activities (resource constraints).

A solution $S$ of the previous problem is a complete assignment to the activity start times that satisfies all the temporal and all the resource constraints. Given a metric $m$, the optimization version of the PSP is to find a feasible schedule that optimizes the value of $m$.

The representation of a $n$-activities problem $P$ is often realized through a graph $G_P(\mathcal{V}_P, \mathcal{C}_P)$, where the set of nodes $\mathcal{V}_P = \mathcal{V} \cup \{a_0, a_{n+1}\}$ is composed of all the activities that pertains to $P$ plus two dummy activities representing the origin $a_0$ and the horizon $a_{n+1}$ of the scheduling temporal axis, and the set $\mathcal{C}_P$ contains the temporal constraints insisting between pairs of activities in $P$. Each edge $(a_i, a_j) \in \mathcal{C}_P$ in the graph representing a time lag between activities $a_i$ and $a_j$, is labeled with an interval $[l_{ij}^{min}, l_{ij}^{max}]$ which quantifies the minimum ($l_{ij}^{min}$) and the maximum ($l_{ij}^{max}$) distance permitted between the activities. According to this representation, a solution to the scheduling problem $P$ can be formulated by adding to the graph $G_P$ a set $\mathcal{C}_S$ of simple precedence constraints $a_i \prec a_j$ (solution constraints). Each solution constraint aims at separating all the pairs of activities that cause a resource conflict by competing for the same resource beyond the resource maximum available capacity.

Very informally, a schedule under execution at time $t = t_E$ can be defined as a partition $\mathcal{V}(t_E) = \mathcal{V}_{\prec t_E} \cup \mathcal{V}_{\approx t_E} \cup \mathcal{V}_{\succ t_E}$ where $\mathcal{V}_{\prec t_E}$ is the set of the activities that have already terminated, $\mathcal{V}_{\approx t_E}$ is the set of the activities currently under execution, and $\mathcal{V}_{\succ t_E}$ is the set of the activities that have yet to begin.

Before producing a formal definition of the environmental uncertainty that permeates the execution of a schedule, we need to introduce the following concept of *Instant Modifier*:

**Definition 2** (*Instant modifier*) Let $P = \langle \mathcal{V}, \mathcal{C}, \mathcal{R} \rangle$ be a scheduling problem and $S_k \in \mathcal{S}(P)$ be one of its solutions under execution at the instant $t = t_E$ (*Time of Execution*): an Instant Modifier $mod^Z(t)$ is an operator whose application $mod^Z(t_E) * P$ on the problem $P$, produces an alteration of the component $Z \in \{\mathcal{V}, \mathcal{C}, \mathcal{R}\}$ to be applied at time $t = t_E$.

We can now exploit the notion of Instant Modifier to give an alternative definition of solution $S$:

**Definition 3** (*Solution of a scheduling problem*) Let $P = \langle \mathcal{V}, \mathcal{C}, \mathcal{R} \rangle$ be a scheduling problem: a consistent solution $S_k \in \mathcal{S}(P)$ of the problem $P$ can be defined as follows:

$$S_k = mod_{\star}^{C,k} * P$$

where $mod_{\star}^{C,k}$ is the instant modifier that integrates the original constraint set $\mathcal{C}$ with the solution constraints set $\mathcal{C}_S = \{c_1^S, c_2^S, \ldots, c_r^S\}$:

$$\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_S$$

As stated in the previous definition, both the scheduling problem instances $P$ and their relative solutions $S \in \mathcal{S}(P)$ *share the same structure*. In fact, both can be formalized through a tuple $\langle \mathcal{V}, \mathcal{C}, \mathcal{R} \rangle$, the difference being in the number and/or allocations of the constraints $c_i \in \mathcal{C}$ that are necessary to eliminate the temporal and/or resource conflicts present in $P$.

The analysis of the base problem's structure and main features is essential to derive a meaningful characterization of the associated reactive scheduling problem. All this considered, we provide the following definition:

**Definition 4** (*Reactive scheduling problem—RSP*) Given a base scheduling problem $P_{base} = \langle \mathcal{V}, \mathcal{C}, \mathcal{R} \rangle$, the *Reactive Scheduling Problem* can be formalized as a tuple $\langle P_{base}, \Delta_{\mathcal{V}}(t), \Delta_{\mathcal{C}}(t), \Delta_{\mathcal{R}}(t), t_0 \rangle$ where:

- $\Delta_{\mathcal{V}}(t) = mod^{\mathcal{V}}(t) * P_{base}$ is the set of modifications applied to $\mathcal{V} = \{a_1, \ldots, a_n\} \in P_{base}$ by the modifier $mod^{\mathcal{V}}(t)$, at time $t$;
- $\Delta_{\mathcal{C}}(t) = mod^{\mathcal{C}}(t) * P_{base}$ is the set of modifications applied to $\mathcal{C} = \{c_1, \ldots, c_k\} \in P_{base}$ by the modifier $mod^{\mathcal{C}}(t)$, at time $t$;
- $\Delta_{\mathcal{R}}(t) = mod^{\mathcal{R}}(t) * P_{base}$ is the set of modifications applied to $\mathcal{R} = \{r_1, \ldots, r_n\} \in P_{base}$ by the modifier $mod^{\mathcal{R}}(t)$, at time $t$.

A solution $S_{exec}$ of the RSP is a solution to the new Scheduling Problem $P_{exec}$ defined by the tuple $\langle \mathcal{V} + \Delta_{\mathcal{V}}(t), \mathcal{C} + \Delta_{\mathcal{C}}(t), \mathcal{R} + \Delta_{\mathcal{R}}(t) \rangle$, that is:

$$S_{exec} = mod_{\star}^{C,k} * P_{exec}$$

The reader should notice that solving the RSP is different from solving an ordinary PSP, because of the presence of the time variable $t$. Being the RSP a dynamic problem, the time component plays an essential role in the solving process of the RSP instances. For the time being, it is important to highlight that, given a base scheduling problem $P_{base}$ and three modification sets $\Delta_{\mathcal{V}}(t)$, $\Delta_{\mathcal{C}}(t)$, $\Delta_{\mathcal{R}}(t)$, the following tuples $\langle P_{base}, \Delta_{\mathcal{V}}(t), \Delta_{\mathcal{C}}(t), \Delta_{\mathcal{R}}(t), t_1 \rangle$ and $\langle P_{base}, \Delta_{\mathcal{V}}(t), \Delta_{\mathcal{C}}(t), \Delta_{\mathcal{R}}(t), t_2 \rangle$ $(t_1 \neq t_2)$, represent two different RSPs, as the mere passing of time inherently modifies the activity reallocation opportunities in case a rescheduling is necessary. Intuitively, a rescheduling action triggered by the occurrence of an exogenous event, may yield different results depending on the number of activities that have begun the execution or that have already terminated, and such number is obviously a function of time.

We now provide the definition of the optimization version of the Reactive Scheduling Problem, as it reveals more interesting for our purposes:

**Definition 5** (*Reactive scheduling optimization problem—RSOP*) Given a base optimization scheduling problem $P_{base}^{opt} = \langle \mathcal{V}, \mathcal{C}, \mathcal{R}, m, \mathcal{O} \rangle$, with $O \in \{\min, \max\}$, and a solution $S_{base}^{opt}$ that optimizes the value of the objective function $m$, the *Reactive Scheduling Optimization Problem* can be formalized as the tuple $\langle P_{base}^{opt}, \Delta_{\mathcal{V}}(t), \Delta_{\mathcal{C}}(t), \Delta_{\mathcal{R}}(t), m, t_0 \rangle$ (see Definition 4), where the solution $S_{exec}^{opt}$ of the RSOP is one that optimizes the value of $m$.

## Measuring the scheduling efficacy

In order to measure how well different scheduling procedures perform against executional uncertainty, it is necessary to establish the criteria (i.e., the function $m$ in Definition 5) with respect to which such evaluation will be carried out. Unfortunately, given the complexity of the problem, there is no unique way to generally define the effectiveness of a scheduling procedure: generally, an "ideally robust" schedule should be able to instantly absorb all kinds of events, delays in the activity start times, changes in activity durations, resource breakdowns, etc, without any consistency and quality loss.

*Defining schedule robustness* It is commonly acknowledged that no unique definition of solution robustness can be extracted from literature: the most suitable definition normally depends on the particularly considered perspective. For instance, according to (Leon et al. 1994; Leus and Herroelen 2004), schedule robustness can be considered as the ability to generally preserve makespan quality. In (Sevaux and Sörensen 2002) instead, the authors pick up two different

aspects of robustness, one related to solution quality and the other related to solution *continuity*.[1]

In (Ginsberg et al. 1998) the concept of robustness is mostly studied from a dynamic point of view, where the attention is focused on the ability of the solution to keep up with the execution pace: the aspects related to solution continuity are still taken into account and much emphasis is given to robustness as the capability to minimize the necessary disruptions of the evolving solution. Yet, schedule reactiveness is one of the authors' great concerns, as robustness is assessed also as a function of the time needed to synthesize alternative solutions.

In the light of what precedes, the following three main aspects outstand as particularly contributory to the definition of Robustness:

- *Solution reactiveness*: the aspect related to fast responsiveness to changes: in all cases, a highly desirable feature. The faster the schedule is able to adapt to the new environmental conditions, the lower the possibility of schedule quality degradation during the execution. Keeping up with the execution pace imposed by the environment is essential: failing to come up with an alternative solution in due time, definitely implies execution failure;
- *Solution stability*: the aspect related to continuity maintenance. Given an unexpected event, the higher the schedule's stability, the lower the disruption with respect to the the initial solution's structure.
- *Solution quality*: independently of the specific metric employed for quality evaluation, maintaining such measure as close as possible to the baseline solution's value during the execution is of great importance. For example, in the execution of a plan it is essential to keep the schedule's makespan very close to the original value (increases of the total completion time are often related to the payment of penalty fees that may exceed the project budget constraints).

Unfortunately these characteristics are often mutually conflicting. For instance, a high quality solution is normally computed at a high computational cost; if fast responsiveness to changes is required, in case of a revision it is very probable that much of the solution's quality will be sacrificed for speed. As another slightly more counterintuitive example, also quality and stability can conflict with each other: let us suppose that makespan is chosen as the quality measure; as a consequence of an exogenous event, it can happen that makespan minimization might only be obtained at the cost of a complete schedule re-shuffling, with complete loss of solution continuity.

[1] i.e., the capability to exhibit minimum deviations with respect to the original solution, as exogenous perturbations occur.

*Assessing schedule robustness* Each robustness component can be evaluated through a variety of metrics that cover Reactiveness, Stability and Quality aspects. Given a scheduling problem instance $P$ composed of $n$ activities, examples of such metrics are presented below:

- *Reactiveness*: the technology we employ allows us to rely on the general concept of *temporal flexibility* in order to guarantee fast reschedulings. Such flexibility can be estimated through the *flex* (Aloulou and Portmann 2003) and/or *fldt* metric (Cesta et al. 1998):

$$flex = \sum_{i=1}^{n} \sum_{\substack{j>i \wedge \\ a_i \not\prec a_j \wedge a_j \not\prec a_i}}^{n} \frac{2}{n(n-1)} \tag{1}$$

$$fldt = \sum_{i=1}^{n} \sum_{j=1 \wedge j \neq i}^{n} \frac{slack(a_i, a_j)}{H \times n \times (n-1)} \times 100 \tag{2}$$

where in (2), the $slack(a_i, a_j)$ is the width of the allowed distance interval between the end time of activity $a_i$ and the start time of activity $a_j$. The choice of these metrics is justified by the fact that, given a schedule $S$ modeled as a temporal graph, the *flex* and *fldt* values are directly proportional to the number of different temporal solutions "captured" by $S$. Since stepping from one solution to another in the graph can be performed through propagation algorithms that operate in polynomial time, high *flex* and *fldt* values should guarantee fast responsiveness.

- *Stability*: the capability to preserve the continuity of a solution, that is, to minimize (a) the extent of the effects produced by a perturbation, and (b) the number of activities that are affected by such perturbation, is best accounted for through the *disruptibility* metric (*dsrp*) (Policella et al. 2004b):

$$dsrp = \frac{1}{n} \sum_{i=1}^{n} \frac{slack_{a_i}}{num_{changes}(a_i, slack_{a_i})} \tag{3}$$

The rationale of the previous formula is based on the assumption that, given the ratio $r$ between the *temporal slack* value ($slack_{a_i}$) associated to each schedule activity $a_i$, and the number of activities affected by a disturbance of maximum size $slack_{a_i}$ on $a_i$, the higher is the value of $r$, the lower is the disrupting potential of an exogenous event on the solution.

- *Quality*: the quality of a schedule can be represented by all kind of metrics, depending on which are the aspects of interest that pertain to the particular scheduling application. Given a schedule composed of $n$ activities, one of the measures that traditionally have mostly been employed to represent schedule quality is *makespan* (*mksp*):

$$mksp = max\{c_i\} \quad (i = 1, \ldots, n) \tag{4}$$

where the quantity $c_i$ represents the completion time of activity $a_i$.

*Preserving schedule performances* Schedule Robustness is the quantity with respect to which the scheduling procedures normally optimize. For instance, one might be interested in solving scheduling problems, aiming at fluidity maximization in order to get high solution responsiveness in case of a rescheduling: in these cases, the metric in Eq. 2 would be used as the *m* function to guide the optimization process (see Definition 5). Yet, despite all the optimization efforts, environmental uncertainty might still spoil the solution's desired characteristics; therefore, the latter require to be continuously confirmed at execution time. In fact, while high *flex* or *fldt* values can undoubtedly be taken as a reasonable warranty of fast rescheduling performances and high *dsrp* values guarantee a good solution stability, the only way to certify the validity of such behavioral expectations passes through the direct assessment of the quantities of interest, assessment that has to be performed on the executing schedule, as part of the on-line analysis.

The real efficacy of any scheduling approach can therefore be evaluated by periodically keeping track of the *actual* behavior of a solution under execution; through the use of a set of suitable evaluation metrics, it is possible to monitor if, under what conditions, and to what extent, a good estimated value related to a desired characteristic really translates into a correspondingly good actual behavior.

| | Estim. (static) | Eval. (dynamic) |
|---|---|---|
| Reactiveness | *flex, fldt* | CPU time |
| Stability | *dsrp* | *snty* |
| Quality | *mksp* | $\Delta_{mksp}$ |

The previous table presents one possible choice for the metrics that perform online robustness assessment. The evaluation on Reactiveness can be trivially carried out through direct measurements on the CPU time required to perform the reschedulings, the idea being: if a remarkable increase in processing time should be detected on successive reschedulings, greater attention should be paid in the maximization of the *flex* or *fldt* values, even if this may entail a worsening of other significant aspects.

Solution Stability can be evaluated through the following metric, that we called *sensitivity* (*snty*):

$$snty = \sum_{i=1}^{N} \frac{|st_f(a_i) - st_0(a_i)|}{N} \tag{5}$$

where $st_0(a_i)$ and $st_f(a_i)$ represent the start times of the activity $a_i$, computed respectively before and after the revision process. The metric is directly proportional to the disruption level suffered by a solution, after the occurrence of one or more exogenous events. Again, when the *snty* increases during the execution, this information might be used to opt for rescheduling strategies that privilege the increasing of the *dsrp* value. Lastly, given the Quality $q$, dynamic evaluation can be normally carried out by computing the differences $\Delta_q$.

This validation process can be used to gather the necessary information to perform dynamic swaps of different rescheduling strategies, depending on the actual status of the execution. Hence, the motivations of a dynamic confirmation do not only stem from the necessity to assess the validity of the given static estimation measures: they also stem from the need to study, and possibly keep under control, the complex interactions that exist among the different robustness-related aspects, for a proper schedule management.

## Simulating uncertainty during execution

The uncertainty aspects which normally permeate real working environments can be taken into account through the concept of *instant modifier* (Definition 2), which can be considered as the template representation of an executional event. *The idea we pursue is to model the exogenous events through proper instantiations of such templates, and synthesize each reactive scheduling benchmark instance as a sequence of such instantiations.*

### Definition of the different exogenous events

In order to define a benchmark set for the reactive scheduling problem, we refine here the concept of *Instant Modifier* ($mod^Z(t)$) into a number of event types, each modeling a particularly meaningful aspect of environmental uncertainty. Every instant modifier entails an alteration on the $Z$ component of the base scheduling problem (see Definitions 1, 2); depending on the particular nature of $Z$, each modification directly translates into a different type of exogenous event, of which we provide a detailed parametric definition. Every event type is characterized by a set of parameters: the values of all the parameters involved in the following event templates are generated randomically[2] during the synthesis of the benchmarks.

Following is the list of the possible event types, all of them characterized by the $t_{aware}$ parameter, which represents the time of occurrence of the contingency (see Example 1):

– *activity delay* ($e_{delay}$):

$$\langle a_i, \Delta_{st}, t_{aware} \rangle \equiv mod^C(t) \tag{6}$$

[2] Provided the limitations described in the following sections.

this event is modeled as a temporal constraint of extent $\Delta_{st}$ on $a_i$'s start time;

– *growth of activity processing time* ($e_{dur}$):

$$\langle a_i, \Delta_{dur}, t_{aware} \rangle \equiv mod^{\mathcal{C}}(t) \quad (7)$$

this event is modeled as a temporal constraint of extent $\Delta_{dur}$ between $a_i$'s start and end times;

– *lowering of resource availability* ($e_{res}$):

$$\langle r_j, \Delta_{cap}, st_{ev}, et_{ev}, t_{aware} \rangle \equiv mod^{\mathcal{R}}(t) \quad (8)$$

this event defines a reduction of extent $\Delta_{cap}$ of resource $r_j$'s maximum capacity $C_j^{max}$, in the temporal interval $[st_{ev}, et_{ev}]$;

– *variations in the number of activities* ($e_{act}$):

$$\langle f_a, a_k, \overline{req_k}, dur_k, est_k, let_k, t_{aware} \rangle \equiv mod^{\mathcal{V}}(t) \quad (9)$$

this event defines the addition/removal of activity $a_k$;

– *change in the mutual ordering of the activities* ($e_{causal}$):

$$\langle f_c, a_{prev}, a_{succ}, d_{min}, d_{max}, t_{aware} \rangle \equiv mod^{\mathcal{C}}(t) \quad (10)$$

this event defines the addition/retraction of a causal constraint of extent $[d_{min}, d_{max}]$ between the activities $a_{prev}$ and $a_{succ}$.

In the operational use of the RSP benchmarks, a schedule is supposed to be executed and constantly checked for consistency every time a new benchmark instance (event) is injected. The instant at which the *execution simulator system* is supposed to acknowledge the event occurrences is specified through the $t_{aware}$ parameter. The reader should not mistaken the event occurrence time defined by the parameter $t_{aware}$ with the temporal instant which involves the same event along the schedule time-line: the former defines "when" the event will take place; the latter defines "where" the event is localized.

*Example 1* Given the schedule in Fig. 2 and the two following events:

– $e_{dur} = \langle a_2, 6, 7 \rangle$
– $e_{delay} = \langle a_1, 5, 13 \rangle$

the first event occurs at time $t_{aware} = 7$ but is localized at $t_{loc} = 28 + duration(a_2)$; the second event occurs at time $t_{aware} = 13$ but is localized at $t_{loc} = 21$; the temporal distance $t_{loc} - t_{aware}$ (computation time window) reveals a good evaluation of the urgency of an exogenous event.

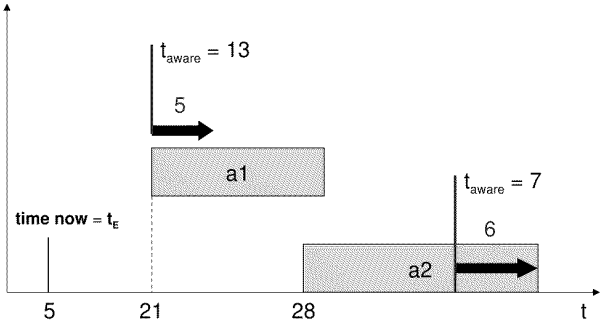


**Fig. 2** Sequential injection of two temporally spaced events

Generating benchmarks for the reactive problem

Figure 3 shows the basic elements of an empirical framework for schedule execution. The framework is composed of the following modules: (1) a Predictive Scheduler, in charge of solving the static sub-problem by synthesizing the baseline schedule; (2) the Reactive Scheduler, which receives in input the baseline schedule and solves the dynamic sub-problem by taking care of the solution maintenance during the execution; (3) the Testset Generator module, which produces the reactive scheduling benchmark sets. In this section we discuss some issues that have been tackled during the design of the Testset Generator module for the reactive scheduling problem.

As Fig. 3 shows, each reactive scheduling benchmark set (composed of exogenous events) is treated as a second input to the Reactive Scheduler, which is called to acknowledge the events and revise the executing schedule according to the updated conditions. Note that the PSP instances are a necessary input for Testset Generator; in fact, the knowledge of the structure of each base scheduling problem instance $P$ is necessary to synthesize exogenous events that are meaningfully related to $P$. Besides, this is in perfect accordance with Definition 2 of Instant Modifier, presented as an operator that applies to a problem instance $P$ in order to alter some of its original properties.

The base scheduling problem we focus upon in this work is the RCPSP/max problem (resource constraint project scheduling problems with minimum and maximum time windows, (Bartusch et al. 1988)), even though for the following

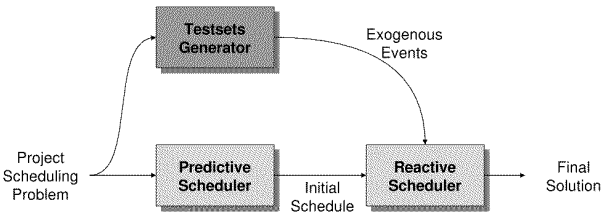


**Fig. 3** Reactive scheduling benchmark generation path

analysis we will exclusively focus on its temporal aspects, disregarding any resource constraints (Simple Temporal Problem, or STP Dechter et al. 1991). Depending on the network structure of each problem instance and on the values pertaining to the network constraints, every time point $tp_i$ is associated to an interval $[lb_i, ub_i]$ of admissible values for $tp_i$. The width $ub_i - lb_i$ of each interval is inversely proportional to the "constrainedness" of the associated time point; Obviously, the highest level of time point constrainedness is reached when $ub_i = lb_i$, in which case only one feasible value exists for $tp_i$.

*Generating consistent events* The knowledge of the temporal bounds that affect every time point is essential to produce events of meaningful size. Benchmark generation is in fact a random process: recognizing the precise limits within which to perform such randomization helps synthesizing events *that do not uselessly overstress the network structural properties*. This is why an aprioristic analysis of the initial temporal network underlying each scheduling problem instance is necessary.

Given a scheduling problem $P$ and the related temporal problem STP, the delay $\Delta_{st}$ in Eq. 6 will therefore be randomly chosen inside the interval $[0, ub(st_i) - lb(st_i)]$, while the value $\Delta_{dur}$ in Eq. 7 will be randomly chosen inside the interval $[0, ub(et_i) - lb(st_i) - p_i]$ (where $p_i$ is $a_i$'s processing time).

Yet, choosing the event size between the previous bounds represents a necessary *but not sufficient* condition to guarantee event feasibility. In fact, during the execution of the schedule, the constrainedness of the current solution's temporal network tends to constantly increase;[3] since the previously defined bounds are based on the analysis of $P$'s network (which is characterized by the lowest constrainedness value), and the sizes of all the exogenous events is computed on the basis of such bounds, the produced benchmark cannot be guaranteed to model a consistent situation for all possible executions. Nonetheless, a wise management of temporal constrainedness can be profitably used to control the benchmark difficulty, as shown in the next paragraph.

*Measuring the difficulty of the events* As explained in section "Measuring the scheduling efficacy", different metrics can be taken into account to assess the schedule's robustness, depending on the particular executional aspect of interest. According to our view, the gravity of the synthesized events can be determined through the same metrics $\mu$ used to measure schedule's robustness: each event will aim at interfering with the aspect measured by $\mu$, and the extent of the event is determined by the variation in $\mu$'s value caused by the introduction of the event. *The idea is to measure the difficulty of a set of events $\mathcal{E} = \{e_1, \ldots, e_n\}$ on the basis of the extent of the modifications that these events introduce in the scheduling problem's structure.*

Let us consider a scheduling problem $P^k$ obtained by introducing the event $e_k$ during the execution; given a generic metric $\mu$, it is possible to compare the structural properties of the problem $P^k$ with respect to the previous problem $P^{k-1}$, and therefore assess the event difficulty, in the following ways:

- by measuring the absolute variation with respect to the previous problem:

$$\Delta_\mu = |\mu(P^k) - \mu(P^{k-1})| \tag{11}$$

- by measuring the speed of this variation:

$$\frac{\Delta_\mu}{\Delta_t} = \frac{|\mu(P^{k+1}) - \mu(P^k)|}{\Delta_t} \tag{12}$$

  where $\Delta_t$ represents the temporal distance between $e_{k+1}$ and $e_k$.

Equation 12 measures how the events are spaced over the horizon: given two events, the closer they are, the more critical the situation (this corroborates the necessity to define $t_{aware}$ values which are solution independent).

In RCPSP/max problems there are two aspects of primary importance, the temporal and the resource aspect; depending on the chosen metric, it is possible to take into account both. Some temporal metrics have already been introduced in section "Measuring the scheduling efficacy"; they represent a particular interpretation, at the activity symbolic level, of the metric previously defined as *constrainedness*. Such generic temporal information can be further specialized into measures describing specific aspects of the solution.

In order to understand the bias produced by a set of events on the resource-related characteristics of the scheduling problems, we employ another well-known measure, namely the *Resource Strength* (Schwindt 1998), defined as follows:

$$RS_k = \frac{C_k^{max} - r_{min}^k}{r_{max}^k - r_{min}^k} \tag{13}$$

where, given the resource $r_k$, $r_{min}^k = \max_{i=1..n} req_{ik}$, is the maximum usage of resource $r_k$ by any activity, while $r_{max}^k$, is the peak demand of resource $r_k$ computed on the early start time solution of the infinite capacity version of the problem.[4]

For each resource $r_k$, this measure takes into account the resource availability level with respect to the task requirements. The metric $\mu$ we are interested at, is based on the

[3] Under the current hypothesis that the original problem's constraints are never retracted and that the exogenous events are modeled as further constraints.

[4] The infinite capacity version of a scheduling problem $P$ is obtained by relaxing all the resource constraints in $P$.

average value of $RS_k$ over all the resources employed in the scheduling problem $P$:

$$RS_P = \frac{\sum_{k=1}^{m} RS_k}{m} \tag{14}$$

We can use the metric $RS_P$ to generate events that controllably reduce the "safety margin" between the resource maximum capacities and the average resource utilization, thus increasing the probability (if not directly causing the occurrence) of contention peaks.

## A framework for schedule dynamic analysis

Analyzing how the effectiveness of the reactive scheduling procedures may be affected by the proactive strategy used to synthesize the baseline solution is in our opinion as important as being able to understand the implications of assess the best baseline schedule production strategy on the base of the desired dynamic characteristics of the solution.

For this reason, we have devised an experimental platform which enables us to compare different approaches to schedule synthesis and execution in a fair and controlled way. The platform we describe in this section is designed to be modular and reusable, and is the same that will be used to carry out the experimental analysis which is the object of the final part of this work.

Figure 4 shows the complete schema of the platform. The architecture is composed of three blocks: the *Solver* and the *Event Generator* work off-line and have the job of, respectively, computing the initial solution (baseline schedule), and generating the exogenous events intended to disturb the schedule execution (see section "Generating benchmarks for the reactive problem").

The third block, the *Schedule Execution Control* module, works on-line, and is responsible of performing a complete *Execution Instance* of the scheduling solution, that is, a simulation of the execution of the initial solution $S$ until total schedule completion or until a failure is encountered. In summary, this module accepts three inputs:

(1) the problem instance $P \equiv G_P(V_P, E_P)$;
(2) the solution constraint set $E_S$ (which, together with $P$ compose the baseline solution $S \equiv G_P(V_P, E_P \cup E_S)$);
(3) the events synthesized by the *Event Generator*, necessary to model the environmental uncertainty.

The disturbing events are injected during the simulated execution at times specified by the $t_{aware}$ parameter, and their effects are counteracted by the execution module, which is endowed with a portfolio of *rescheduling* algorithms to the aim of restoring and maintaining schedule consistency whenever necessary.

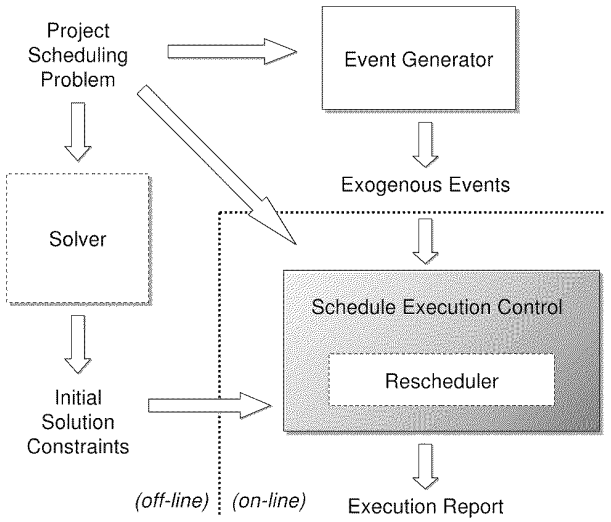


**Fig. 4** The complete schema of the schedule dynamic analysis framework, subdivided in the off-line and on-line section

The *Schedule Execution Control* module simulates the advancement of time and is responsible for dispatching the activities according to their earliest start time in the underlying Simple Temporal Network (STN). When a unexpected temporal event is acknowledged, the STN is modified accordingly, and the relative "updated problem" is passed to and reasoned upon by the *Rescheduler*, which eventually produces an "updated solution". The latter will therefore be taken as the current baseline solution managed by the Progress Monitor to guarantee the continuation of execution.[5]

## Designing an experimental analysis

In this section we discuss the design of the experimental analysis which results are presented in section "Experimental results and discussion". All initial schedules have been produced using two off-line procedures characterized by different degrees of proactiveness, while the on-line management has been demanded to two reactive scheduling procedures whose rescheduling action is characterized by a different level of invasiveness.

### Selecting the off-line strategies

For the present analysis, we have chosen two particular off-line strategies, both following the Precedence Constraint Posting (PCP) approach. Both strategies produce solutions

[5] It is worth noting that the execution can fail for two different reasons: (a) because the exogenous event "overstresses" the solution's temporal structure, and (b) because the adopted rescheduling procedure fails in finding an alternative solution.

that retain a certain degree of temporal flexibility (i.e., robustness); yet, each solution type is characterized by different structural properties. The solutions produced with the first strategy are called *flexible schedules* ($\mathcal{FS}$), while the second strategy returns solutions called *partial order schedules* ($\mathcal{POS}$).

$\mathcal{FS}$ solutions are computed using the ISES algorithm, a randomized optimization version of a PCP greedy algorithm (Cesta et al. 1999); in (Cesta et al. 1998) it is also shown that though the solutions returned by the ISES algorithm are inherently characterized by a certain level of resilience at execution time due to the flexibility of the time points, they represent resource consistent solutions only if specific values from the time points admissibility intervals are chosen, as described in the following definition of *Flexible Schedule*:

**Definition 6** (*Flexible schedule*) A flexible schedule for a problem $\mathcal{P}$ is a network of activities, (readily interpretable as a temporal graph), such that a feasible solution for the problem is obtained by allocating each activity at the temporal lower bound allowed by the network (Earliest Start Time Solution).

In order to cope with the previous limitation which characterizes the flexible schedule, a generalization of the STN produced by the PCP phase is proposed in (Cesta et al. 1998; Policella et al. 2004a), which aims at defining a set of both time and resource feasible solutions. In the cited works, the authors consider the generation of temporally flexible schedules by focusing on the construction of activity networks, each representing a set of solutions of the scheduling problem, *that are consistent either from the temporal and from the resource standpoint*. This new representation is called *partial order schedule*, and can be defined as follows:

**Definition 7** (*Partial order schedule*) A partial order schedule $\mathcal{POS}$ for a problem $\mathcal{P}$ is an activity network, such that any possible temporal solution is also a resource-consistent assignment.

In this work, we have re-implemented the algorithms of (Policella et al. 2004a) to synthesize the $\mathcal{POS}$s starting from the $\mathcal{FS}$ solutions and applying to such solutions the *Chaining* process (see Fig. 5).

Selecting the on-line strategies

Two on-line strategies are chosen, which differ in the level of rescheduling invasiveness with respect to the schedule's structure. We address the issue of local versus global rescheduling as this represents an aspect that strongly affects dynamic behavior during schedule execution. Rescheduling is performed using computationally lighter versions of the constraint-based algorithms used off-line, which are global in

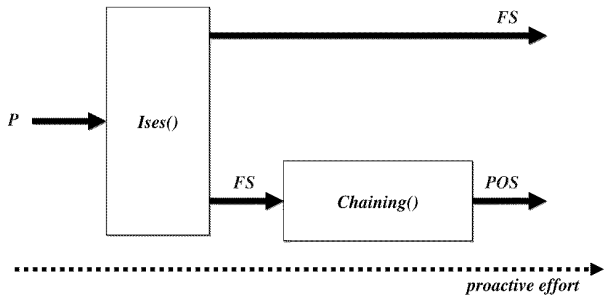


**Fig. 5** Partial order schedules as an improvement of the flexible schedules

nature. Local/global behavior is obtained by properly tweaking on the constraints belonging to the updated problem that the scheduler is called to reason upon, therefore forcing the rescheduling algorithms to act more or less locally. More specifically, we designed the following two basic on-line strategies:

- a *global reactive strategy* which, before calling a PCP procedure in order to eliminate the conflicts caused by the occurrence of an exogenous event, removes from the current solution all the constraints imposed by the previously performed solving process. We call this a *Retraction strategy*;
- a *local reactive strategy*, which enacts a more local effect on the current solution, as it never removes any solution constraints possibly imposed by previous reschedulings. We call this a *No-Retraction strategy*.

*Putting it all together* As Fig. 5 shows, building $\mathcal{POS}$ solutions implies a greater proactive effort than producing $\mathcal{FS}$ solutions; similarly, the local reaction strategy represents a smaller reactive effort with respect to the global reaction strategy, as the latter tends to limit the number of activities involved in the possible rescheduling process. The proactive and reactive strategies described above can therefore be used to form four different scheduling combinations: Fig. 6 shows how such combinations can be placed in the ⟨*off-line, on-line*⟩ plane, depending on the proactive/reactive effort implied by the chosen procedure and according to the scheduling interpretation presented in section "Introduction".

Description of the experiments

The experiments consist in simulating the execution of a number of baseline schedules, computed as solutions of known RCPSP/max scheduling benchmark sets; in particular, the *j*100 scheduling problem benchmark has been used (Kolisch et al. 1998), composed of 540 scheduling problem instances where each instance is composed of 100 activities competing for 5 multi-capacity resources. In order to com-

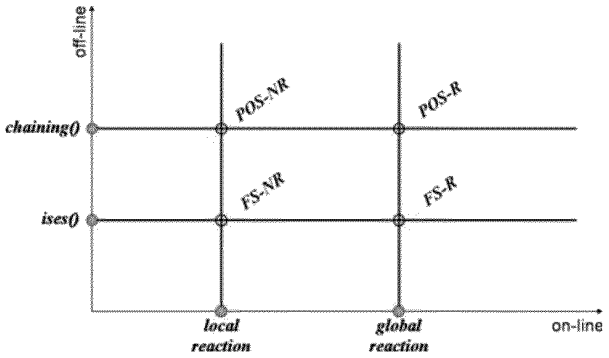


**Fig. 6** Justifying the position of the *local* and *global* reactive procedures on the on-line axis

pare various combinations of proactive and reactive scheduling strategies, all the previous problem instances are solved by either the ISES (FS case) and the CHAINING (POS case) algorithms, as explained in section "Selecting the off-line strategies"; subsequently, each baseline schedule is executed according to a *No-retraction* (NR case) or *Retraction* (R case) reactive strategy, as described in section "Selecting the on-line strategies".

The executions of every baseline solution are disturbed with sets of exogenous events (reactive scheduling benchmark sets) of controlled difficulty. By using the Testset Generator described in section "Generating benchmarks for the reactive problem", we have created four different benchmark sets for every scheduling problem instance, so as to execute every baseline solution under four different simulated environmental conditions of increasing difficulty. In other words, the execution of each solution has been repeated four times, each time being disturbed by a reactive scheduling benchmark set of increasing size.

A further scheduling strategy (fixed-time) has been added, in order to compare the behavior of fixed-time schedules (schedules that retain no temporal flexibility at all), with flexible schedules. The execution of fixed-time solutions has been simulated by forcing a rescheduling on the solution *each time* an exogenous event occurs.

It should be remarked that the four event sets, related to each scheduling problem instance, have been created independently from one another. In other words, given the scheduling problem $P$ and the four event sets $\{S_1^P, S_2^P, S_3^P, S_5^P\}$ associated to $P$, the two sets $S_i^P$ and $S_j^P$ do not have any event in common, $\forall\, i, j \in \{1, 2, 3, 5\}$.

The reactive scheduling benchmark sets have been chosen of exclusively temporal nature, in particular, all the exogenous events sets are composed of delays in the activity start times (6) and/or lengthenings in the activity durations (7).

## Experimental results and discussion

The outcome of the experiments offer vast analytical opportunities; in this section, we survey such results, providing a useful interpretation when necessary, and attempting to underscore the possibly present inter-relations among the obtained data. All the plots we present in this section are organized so as to show how the increasing difficulty of the reactive scheduling problems affect each of the following aspects of interest.

*On-line CPU time requirements* This aspect is measured as the average CPU time spent to perform all reschedulings for all the Successfully Completed Execution Instances, according to the following formula:

$$CPU_{on} = \frac{\sum_{i=1}^{N_{ok}} cpu_{on,i}}{N_{ok}}$$

where $N_{ok}$ is the number of Successfully Completed Execution Instances, and $cpu_{on,i}$ is the total CPU time spent for the reschedulings of the $i$th solution.

Solution *reaction time* is the aspect analyzed here: from Fig. 7a, we immediately witness a difference of behaviors between the local (NR) and global (R) scheduling strategies. The former exhibit better reaction times[6] (the $y$-axis is labeled in milliseconds). Moreover, within the local and the global strategies, the $\mathcal{POS}$s are always quicker than the $\mathcal{FS}$s. This last feature represents an important confirmation of the theoretical expectations that motivated the research study on $\mathcal{POS}$ (Policella et al. 2004a). Moreover, it proves that the two *flex* and *fldt* metrics represent a reliable means for estimating system reactiveness. Indeed, Fig. 7a also underscores that the *locality* effect is stronger than $\mathcal{POS}$'s inherent reactiveness, in lowering the online computational efforts, as the FS-local strategy performs better than the POS-global.

The worst behavior is exhibited by the fixed-time schedules; in fact, given the brittleness of these solutions, a new rescheduling is necessary at every exogenous event occurrence, with heavy repercussions on the CPU time demand.

*Makespan preservation* Fig. 7b shows the schedule makespan deviation ($\Delta_{mk}$), computed between the beginning and the end of all the successfully completed execution instances. This aspect is measured as the average difference between the makespan at the beginning and at the end of every Successfully Completed Execution Instance, according to the following formula:

$$\Delta_{mk} = \frac{\sum_{1}^{N_{ok}} [makespan(S_{ok}) - makespan(S_{init})]}{N_{ok}}$$

The results show again that the local (NR) reaction strategies perform more conservatively than the global (R) strate-

[6] As we will see later, many of the characteristics associated to the various strategy combinations are inter-related.

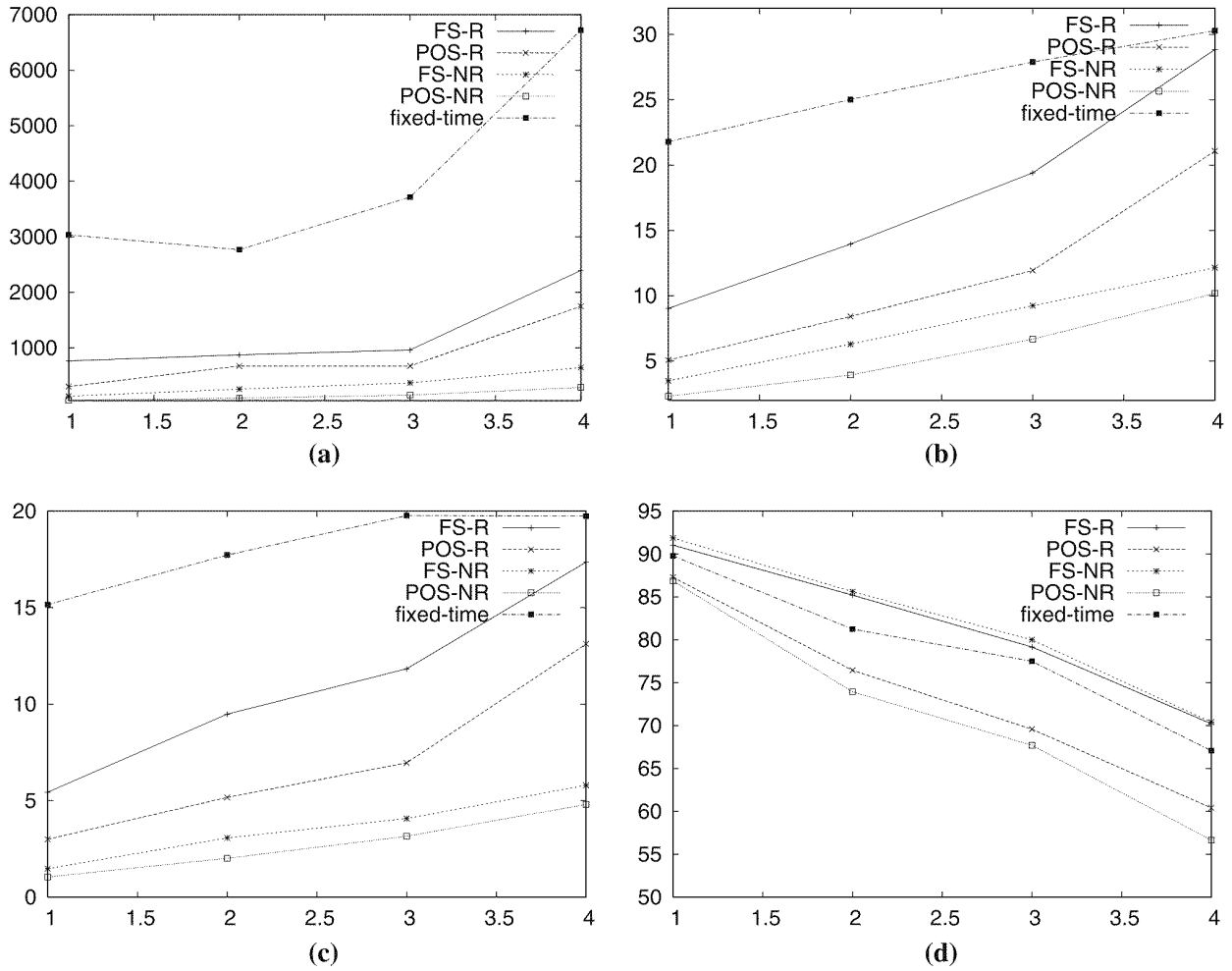


**Fig. 7** Results from the executions of the j100 scheduling benchmark instances; **a** CPU time, **b** makespan deviation, **c** stability deviation, **d** execution success rate

gies, the reason being the following: the No-retraction approaches, acting locally, tend to interfere less heavily with the original structure of the solution.

We also notice that the local strategies are not only the ones which exhibit a better behavior, but they also maintain the flattest derivative with respect to the increasing events (flattest curve trend). Within the local and the global approaches, it should be noted that the $\mathcal{POS}$s perform better than $\mathcal{FS}$s, therefore confirming the two following facts: (1) the CHAINING procedure, through which the $\mathcal{POS}$s are produced, is makespan-preserving, and in some cases also makespan-improving (Policella et al. 2004a); (2) the $\mathcal{POS}$ solutions generally require fewer reschedulings than the $\mathcal{FS}$s (see later analyses), and therefore the original makespan is more likely to be preserved.

Regarding the last point, a few remarks are required: the employed rescheduling algorithms are lighter versions of the algorithms used to synthesize the baseline solutions. Both represent makespan-optimizing procedures, but while the off-line versions try to iteratively improve the solution, the on-line versions stop when the first feasible solution is found, because of the strict execution timing requirements. Therefore, frequent reschedulings during the execution phase tend to spoil the makespan's initial quality, because the dynamic scheduling procedures cannot equal the makespan minimization capabilities of the off-line schedulers. Given the previous choices, the more rescheduling actions are performed, the more the makespan will tend to deviate from the original optimal value.

This point raises a subtle issue, which is worth deepening in order to acquire a better understanding of the presented results. In our experiments we try to maintain schedule *continuity* (i.e., stability, see next subsection) through the local scheduling techniques, by guaranteeing that every new solu-

tion is as close as possible to the previous one, in terms of preservation of the mutual positions among the activities. Moreover, it is clear that *continuity preservation and makespan optimization are generally conflicting objectives*: after the occurrence of exogenous events, the possibility to perform a complete re-shuffling of the activities pays off in terms of makespan minimization, but at the price of a severe continuity disruption. These observations seem to be conflicting with some of the results we present. For instance, one would expect the global strategies (which allow a greater re-shuffling) to return better makespan values with respect to local strategies: indeed, this is exactly what would happen if we decided to give up reaction times and utilize the same algorithms for both off-line and on-line scheduling (as other experiments confirm); but in the present analysis, such expectations are frustrated because, as we have said, the on-line scheduling algorithm often spoils the makespan of the initial schedules.

*Stability preservation* This aspect is measured as the "sensitivity" of activity start time with respect to the execution process, according to the following formula:

$$\psi = \sum_{i=1}^{N} \frac{|st_f(a_i) - st_0(a_i)|}{N}$$

where, for each $N$-activity Successfully Completed Execution Instance $S_{ok}$, $st_f(a_i)$ and $st_0(a_i)$ are, respectively, the start time of $S_{ok}$ and the start time of the related solution at the beginning of the execution.

Figure 7c shows the results about the stability analysis, which confirm the observations made in the previous subsection about solution continuity. More specifically, the expectations are largely confirmed, where we see that the best stability preservation rates are obtained with the local strategies (POS-NR and FS-NR), where a higher stability value corresponds to the lower curves. The reader should notice the strong similarity between the plots showing the makespan deviations and those showing the stability deviation rates, confirming that these two characteristics, which are often mutually conflicting, exhibit instead a converging trend under the current conditions.

*Execution instances success rate* This aspect is measured as the percentage of Successfully Completed Execution Instances ($N_{ok}$), normalized with respect to the number of initially solved problems ($N_{sol}$), according to the following formula:

$$\%\, executed = \frac{N_{ok}}{N_{sol}} * 100$$

By far, one of the most desirable characteristic of a scheduling approach is how well such approach succeeds in guaranteeing a high rate of successfully completed executions. Trivially, having recognized the types of possibly occurring exogenous events, and having evaluated the average difficulty of the reactive scheduling problem (how hard the environmental uncertainty is going to influence the schedule execution), there is little point in assuring very a good theoretical behavior (high reactiveness, high stability, etc.) if the executing schedule is very likely to undergo an early failure during the execution.

We remind that in the present experimental evaluation, a schedule is considered successfully executed if and only if *all* of its activities have successfully come to completion. No retraction of the original constraints is allowed, though this aspect represents an extremely interesting research issue; instead, our analysis is focused on assessing the "endurance" of any given solution when it is under stress, so as to evaluate both the quality of the rescheduling actions and the quality of the proactive off-line procedures. While the former can be evaluated on the basis of how well they succeed in maintaining the solution's original characteristics, the latter can be evaluated on the basis of how well such characteristics help in hedging against uncertainty; obviously this last evaluation cannot be performed if major modifications on the initial problem's structure are allowed.

The result analysis carried out so far has highlighted that many of the expectations on the partial order schedules are actually confirmed, in terms of reactiveness, stability and makespan preservation, etc.: *unfortunately, all the previous good characteristics come at a price*. As evident from Fig. 7d, we observe that all the POS-based strategy combinations exhibit a very low execution success rate, if compared with the FS-based approaches. In particular, the POS-based execution strategies obtain a success rate as low as 56% after 5 events, against a stable 70% rate returned by the FS-based approaches.

Moreover, the figure also shows a slowly increasing trend toward a worsening of this effect, as the number of exogenous events grows. Though this result may seem counter-intuitive it does not represent a surprise; in fact, in general, a $\mathcal{POS}$ solution is more "constrained" than a flexible one, as the constraints generated by the Chaining procedure must necessarily impose a stronger condition on the TCN, as they have to guarantee both temporal and resource feasibility. The experimental results consequently reflect the presence of this trade-off between the ability to respond to changes (flexible schedules), and the ability to provide prompt reply to possible executional changes ($\mathcal{POS}$s).

*Frequency of Reschedulings* This aspect is measured as the percentage of the performed rescheduling actions ($N_{res}$) with respect to the number of injected disturbs ($N_{dist}$), according to the following formula:

$$\%\, resched = \frac{N_{res}}{N_{dist}} * 100$$

So far, we have acknowledged the superiority of the local strategies on the global counterparts. We want now to high-

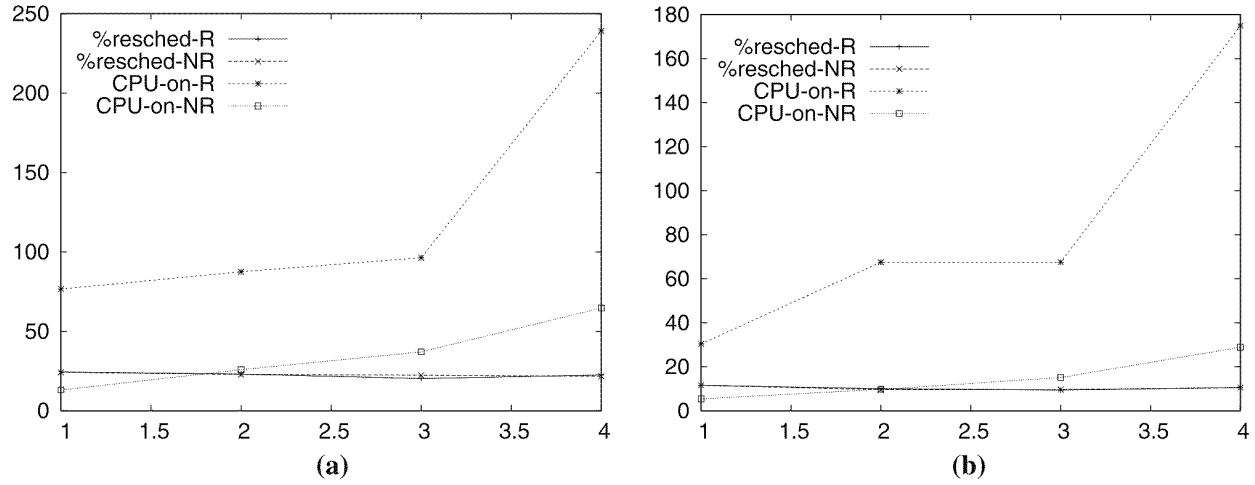


**Fig. 8** Cross-comparing the rescheduling rate with the online CPU time; **a** FS case, **b** POS case

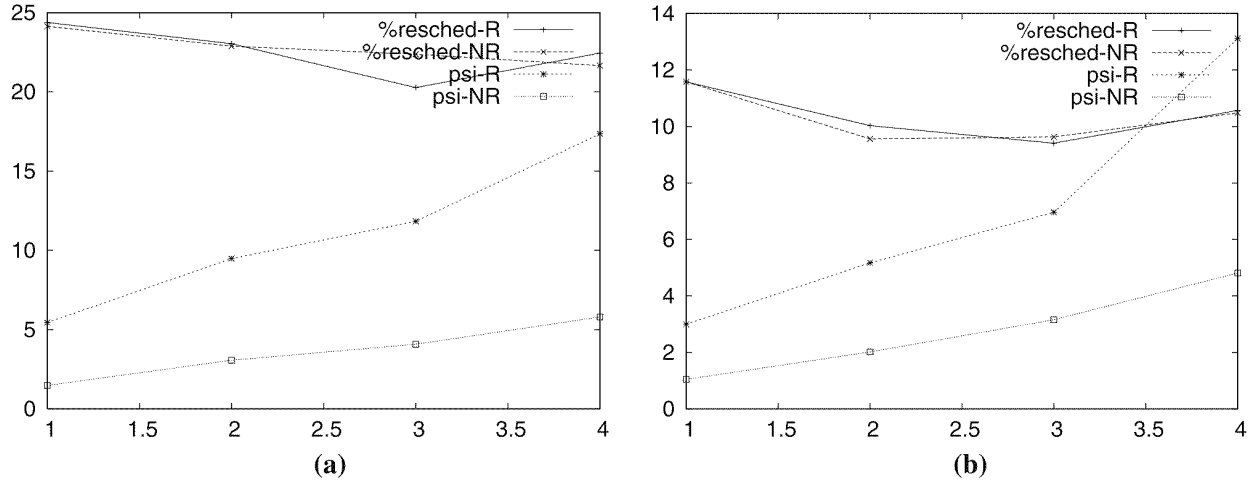


**Fig. 9** Cross-comparing the rescheduling rate with the solution stability rate; **a** FS case, **b** POS case

light an important aspect which weighs a great deal on the dynamic behavior of a schedule: the frequency of the reschedulings. In fact, the number of reschedulings can greatly affect reactiveness, makespan and stability: the fewer the reschedulings, the faster the system reacts, the smaller the related makespan deviations, the more stable the solution.

In order to get a further interesting insights of the experimental data, we perform some cross comparisons of the obtained results. Figure 8 compares the rescheduling rates with the online CPU time, where Fig. 8a and b show, respectively, the data related to the $\mathcal{FS}$ and the $\mathcal{POS}$ execution instances. Both plots are extremely similar, underscoring the fact that the feature we are going to describe does not depend on the particular proactive policy employed. The figures illustrate a peculiar circumstance: despite the comparable rescheduling rates exhibited by both local and global rescheduling policies, the local approach returns much faster reaction times. This result is explained by the fact that the local search explores smaller search spaces, and therefore returns the solutions more quickly. Moreover, it should be observed that in the global approach case, the CPU time requirements are characterized by a much steeper derivative with respect to the local approach case, as the number of exogenous events grows; this last property is valid for both $\mathcal{FS}$s and $\mathcal{POS}$s.

Another interesting cross-comparison involves the rescheduling rate and the solution stability. As Fig. 9 shows, for both the baseline schedule types ($\mathcal{FS}$ or $\mathcal{POS}$), the comparable number of reschedulings between the local and global strategies does not directly translate into a comparable solution stability rate.

Inspecting Fig. 9a, we observe that the stability rate of the local strategy is always higher than for the global strategy, maintaining a flatter and more constant increase trend, regardless of the number of events. Conversely, the stability rate of the global strategy exhibits a hike after three exogenous events. Exactly the same behavior can be observed for the $\mathcal{POS}$ type solutions, in Fig. 9b; these results confirm the circumstance that global reschedulings entail a greater schedule reconfiguration extent with respect to local reschedulings.

## Conclusions

In this paper we extend the definition of scheduling problem to include the dynamic phase of schedule execution, and present a methodology to assess the efficacy of proactive and reactive scheduling methodologies against environmental uncertainty. To this aim, a formal characterization of reactive scheduling problem is provided, as well as a definition of uncertainty, in terms of a set of exogenous event templates of measurable gravity.

A general Schedule Dynamic Analysis framework is presented, designed to be used with different off-line and on-line scheduling algorithms, that can be easily plugged in as interchangeable components. The implemented system allows to perform different sets of reproducible experiments regarding the dynamic behavior of schedules synthesized with various techniques, through a series of simulated executions performed under controllable environmental conditions.

An example of use of the introduced framework is subsequently presented, by simulating the execution of a number of baseline schedules under different and reproducible environmental conditions of increasing gravity, to the aim of assessing the efficacy of the deployed proactive/reactive scheduling approach combinations. As the experimental results show, the performed study reveals extremely useful for the attainment of the following goals: (1) to provide confirmation of the statically presumed characteristics of the baseline solutions, depending on the particular scheduling procedure employed for their production; (2) to discover the possible presence of hidden drawbacks associated to particular scheduling approaches that could have never be found out of a purely static analysis; (3) to provide a *behavioral database* of all the tested solutions, ranked on the basis of (a) the proactive and/or reactive strategy combination respectively employed for their production and/or management, (b) the extent of the stress factor which characterized their execution, and (c) the particular schedule robustness feature of interest.

**Acknowledgements** Amedeo Cesta and Riccardo Rasconi's work is partially supported by MIUR (Italian Ministry for Education, University and Research) under project VINCOLI E PREFERENZE (PRIN), CNR under project RSTL (funds 2007) and ESA (European Space Agency) under the APSI initiative. Nicola Policella is currently supported by a Research Fellowship of the European Space Agency – Human Spaceflight and Explorations Department.

## References

Aloulou, M. A., & Portmann, M. C. (2003). An efficient proactive reactive scheduling approach to hedge against shop floor disturbances. In *Proceedings of 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003)* (pp. 337–362). The University of Nottingham Press.

Aytug, H., Lawley, M. A., McKay, K. N., Mohan, S., & Uzsoy, R. M. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research, 165*(1), 86–110.

Bartusch, M., Mohring, R. H., & Radermacher, F. J. (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research, 16*, 201–240.

Cesta, A., Oddi, A., & Smith, S. F. (1998). Profile Based Algorithms to solve multiple capacitated metric scheduling problems. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems, AIPS-98* (pp. 214–223). AAAI Press.

Cesta, A., Oddi, A., & Smith, S. F. (1999). An iterative sampling procedure for resource constrained project scheduling with time windows. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (pp. 1022–1029). Morgan Kaufmann.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraints networks. *Artificial Intelligence, 49*, 61–95.

El Sakkout, H. H., & Wallace, M. G. (2000). Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints, 5*(4), 359–388.

Ginsberg, M. L., Parkes, A. J., & Roy, A. (1998). Supermodels and robustness. In, *Proceedings of the 15th National Conference on Artificial Intelligence, AAAI-98* (pp. 334–339). AAAI Press.

Herroelen, W., & Leus, R. (2004). Robust and reactive project scheduling: A review and classification of procedures. *International Journal of Production Research, 42*(8), 1599–1620.

Jensen, M. T. (2001). Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing, 1*(1), 35–52.

Kolisch, R., Schwindt, C., & Sprecher, A. (1998). Benchmark instances for project scheduling problems. In J. Weglarz (Ed.), *Project Scheduling—Recent Models, Algorithms and Applications* (pp. 197–212). Boston: Kluwer Academic Publishers.

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

Leus, R., & Herroelen, W. (2004). Stability and resource allocation in project planning. *IIE Transactions, 36*(7), 667–682.

Mc Kay, K. N., Safayeni, F. R., & Buzacott, J. A. (1988). Job-shop scheduling theory: What is relevant? *Interfaces, 18*, 84–90.

Policella, N., & Rasconi, R. (2005). Designing a testset generator for reactive scheduling. *Intelligenza Artificiale, 2*(3), 29–36.

Policella, N., Oddi, A., Smith, S. F., & Cesta, A. (2004a). Generating robust partial order schedules. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming, 10th International Conference, CP 2004, Lecture Notes in Computer Science* (Vol. 3258, pp. 496–511). Springer.

Policella, N., Smith, S. F., Cesta, A., & Oddi, A. (2004b). Generating robust schedules through temporal flexibility. In *Proceedings of the 14th International Conference on Automated Planning & Scheduling, ICAPS'04* (pp. 209–218). AAAI.

Roy, B., & Sussman, B. (1964). *Les problemes d'ordonnancement avec contraintes disjonctives, note DS n. 9 bis*. Paris: SEMA.

Schwindt, C. (1998). A Branch and Bound Algorithm for the Resource-Constrained Project Duration Problem Subject to Temporal Constraints. Tech. Rep. WIOR-544, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Sevaux, M., & Sörensen, K. (2002). A genetic algorithm for robust schedules in a just-in-time environment. Tech. Rep. LAMIH/SP-2003-1, University of Valenciennes.

Smith, S. F. (1994). OPIS: A Methodology and Architecture for Reactive Scheduling. In M. Fox, M. Zweben (Eds.), *Intelligent Scheduling*. Morgan Kaufmann.

Wu, S. D., Beyon, E. S., & Storer, R. H. (1999). A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, *47*(1), 113–124.