Integrating Off-line and On-line Scheduling Approaches

Amedeo Cesta, Nicola Policella and Riccardo Rasconi
ISTC-CNR
Institute for Cognitive Science and Technology
National Research Council of Italy
{name.surname}@istc.cnr.it

Abstract. Solving a scheduling problem consists in finding an initial solution able to model all the complex constraints that relate the problem activities. Moreover, the solving process should be necessarily considered as propaedeutic for the next phase of any schedule's life: its execution in the real world. The inherent unpredictability of real working environments poses serious difficulties to which the scheduling community normally responds by (1) introducing methodologies that tend to increase schedule robustness (off-line phase, proactive approach), and by (2) developing fast-reaction techniques to be employed during schedule execution (on-line phase, reactive approach).

In the previous edition of this workshop we presented a benchmark generator for the reactive scheduling problem. This problem consists in monitoring the execution of a schedule and repairing it every time it is deemed necessary. The main motivation behind our previous work came out from the recognized lack (hence the necessity) of benchmark sets for this specific problem. In this paper we provide an empirical study based on the same benchmark generator which focuses on the mutual interactions among a set of off-line and on-line constraint-based scheduling approaches. We devise a set of closed loop execution management algorithms, and compare their behavior within an experimental framework which allows to directly assess the consequences of each chosen strategy combination, through simulated schedule executions. A number of interesting results are described that open new perspectives for future work based on this integrated schema.

1 Introduction

As the importance of scheduling problem solving methodologies is being increasingly acknowledged by the industry, all the aspects that support automation in the synthesis and management of schedules and that may speed up production lines are receiving constant attention.

One of the most relevant issues regards schedule support at execution time. The dynamism and unpredictability which inherently permeate real-world application domains, make the ability to cope with unexpected events during the schedule execution phase an absolutely primary concern. For instance, the ability to respond automatically (and efficiently!) to unexpected events that occur during normal job-shop floor operations is considered highly desirable. This aspect is being studied in several scientific communities, such as OR and AI, and the relevance of this specific issue is proved by the proliferation of results and surveys [7, 1, 9, 14]. Notwithstanding the relatively recent developments, the problem is still far from being exhausted, and offers much room for investigation.

Traditionally, planning and scheduling communities have tackled the scheduling problem according to one of the two following mainstreams. On one side, much effort has been put into the development of methodologies producing solutions which are characterized by a certain degree of robustness, therefore retaining the ability to absorb the effects of exogenous events (proactive approach). On the other side, the *buffer* that protects the solution against possible disruptions is inherently limited, and the need to devise mechanisms to reactively counteract circumstances that fall beyond its boundaries (reactive approach), is not eliminated.

The present work introduces a schedule management schema where the off-line and on-line approaches are not mutually exclusive. Scheduling is in fact a process where the proactive and reactive phases represent a *continuum*: the task of the scheduler should not be limited to the production of a sequence of activities, as well as the process of controlling schedule executability cannot be exclusively played on the ground of on-line reaction and activity dispatchment. In fact, regardless the proactive approach employed, a dynamic analysis on the actual behavior of the schedule execution is necessary in order to prove, from the operational standpoint, both the efficacy of the choices made and the soundness of the arguments which led to those choices; as for the second point, merely counting on the effectiveness of schedule adjustments at execution time is prone to fostering myopic decisions which may readily result in a complete schedule disruption.

The paper is organized as follows: section 2 presents a detailed description of the particular problem we tackle for our analysis, section 3 briefly explains the structure of the performed experiments, section 4 contains a thorough analysis of the empirical results, while section 5 presents an interpretation of the same results, explaining how the information obtained from this analysis can be used to guide

future research developments.

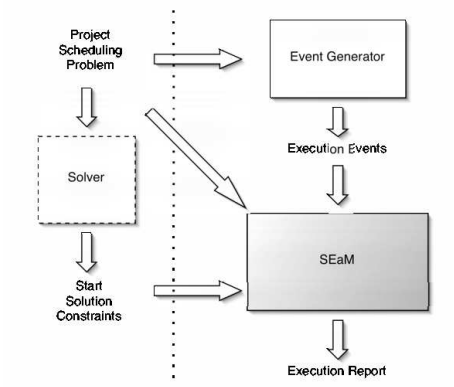

Figure 1: The Scheduling and Execution Monitoring schema

2 The Integration Schema

Analyzing how the proactive phase may influence (and possibly guide) the reactive phase at execution time is in our opinion as important as assessing the best schedule production strategy on the base of the schedule's particular dynamic behavior. The information that can be extracted from the two phases may reveal mutually useful in order to find an optimal strategy combination, as well as the reasons behind its optimality.

To this aim, we have devised an experimental platform which enables us to compare different approaches to schedule synthesis and execution in a fair and controlled way. We use this empirical platform to carry on a set of experiments by simulating the execution of a number of schedules produced with two different proactive methods, disturbing their execution with pre-defined exogenous events, and assessing their behavior by using two separate reactive scheduling policies. The analysis performed in this paper aims at broadening our understanding of one of the most significant and challenging open problems in the Scheduling area: qualitatively and quantitatively unveiling the relation between the structural properties of the initial schedule and the policies chosen for rescheduling, in terms of overall executional behavior.

The scheduling problem we specifically focus upon is the project scheduling problem [3]. The problems of this class are characterized by a rich inner structure: they are based on a network of activities, among which it is possible to identify complex precedence and temporal relations. As a further source of complexity, several heterogeneous resources with different capacities serve the activities according to different modalities.

Having identified this problem, in a recent work [12] we have discussed an approach to the generation of unexpected events to develop reusable benchmarks which could be used to assess the efficacy of re-scheduling policies through *reproducible* experiments.

According to the schedule management schema we base our analysis upon, the off-line solver produces the initial solution and delivers it to the on-line module which takes care of assessing its dynamic characteristics by stressing it in a variety of ways, e.g. by simulating its execution under different environmental conditions. Typically these two aspects are carried on as completely separate tasks, and undoubtedly a very high level of specialization and expertise has been reached in both areas: but how useful would it be if the information yielded by one phase could be immediately available to the other? For instance, a prompt analysis of the dynamical characteristics of a particular solution might reveal invaluable to discover and tweak possible "weak points" of the off-line solving procedure; conversely, different reactive strategies might perform more or less efficiently depending on the chosen off-line approach.

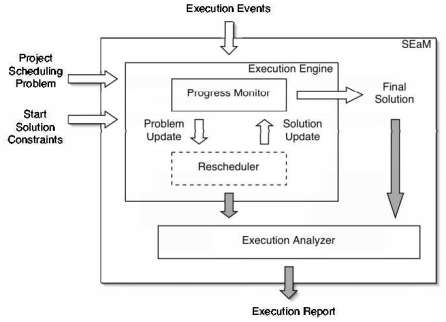

Figure 2: The *SEaM*

In order to fulfill this goal, we have implemented an integrated software tool [13] capable of performing both the static and dynamic stages of schedule management. As Fig. 1 shows, it is composed of three modules: the *off-line solver* and the *Event Generator* work off-line and have the job of, respectively, computing the initial solution and generating the exogenous events intended to disturb the schedule execution; the third module, called *SEaM* (for *Schedule Execution and Monitoring*), is shown in more details in Fig. 2. This module works on-line and is responsible to start, and possibly bring to completion, a simulated execution of the initial solution. A number of disturbing events synthesized by the Event Generator are injected during the simulated execution at specified times, and their effects are counteracted by the *SEaM* on-line solver, which is endowed with a portfolio of *rescheduling* algorithms to the aim of restoring schedule consistency whenever necessary.

By plugging in different versions of either off-line solvers and/or on-line reschedulers it is possible to explore different aspects of the execution problem. In this paper the experimental framework is organized according to the following general conditions: (a) the solvers are chosen so as to produce initial solutions characterized by different levels of temporal flexibility; (b) the designated set of rescheduling algorithms generally follow a constraint-based approach; (c) the exogenous events produced by the *Event Generator* are exclusively of temporal nature (e.g.,

delays in the activities start times, lengthenings in the activities durations, etc.). In the following paragraphs we introduce the scheduling problem we focus on and we describe the previous three points.

Problem specification. We focus on *Project Scheduling* problems [3], composed of the following elements:

- *Activities*. $A = \{a_1, \ldots, a_n\}$ represents the set of activities or tasks. Every activity a_i is characterized by a processing time p_i;
- *Resources*. $R = \{r_1, \ldots, r_m\}$ represents the set of the resources necessary for the execution of the activities. Execution of each activity a_i can require an amount req_{ik} of one or more resources r_k for the whole processing time p_i;
- *Constraints*. The constraints are rules that limit the possible allocations of the activities. They can be divided into two types: (1) the *temporal constraints* impose limitations on the times the activities can be scheduled at; (2) the *resource constraints* limit the maximum capacity of each resource; at no time, the total demand level of any resource being assigned to one or more activities can exceed its maximum capacity.

More specifically, the particular problem we focus upon is the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, or RCPSP/max [2]. This is a particular project scheduling problem which presents constraints that define the minimum and maximum distance between the execution of two activities.[1]

Flexible schedules vs. $\mathcal{POS}$s. We compute an initial schedule with the off-line solver. In contrast to most off-line schedulers, which deliver fixed-time solutions, our approach is based on the general concept of "temporal flexibility" [6]. A temporally flexible solution can be described as a network of activities whose start times (and end times) are associated with a set of feasible values (feasibility intervals). Underlying the activity network there exists a Temporal Constraint Network (TCN [8]), composed of all the start and end points of each activity (time points), bound to one another through specific values which limit their mutual distances (activity on the arc representation). The search approaches used in our schema focus on decision variables which represent conflicts in the use of the available resources; the solving process proceeds by ordering pairs of activities until all conflicts in the current problem representation are removed. This approach is usually referred to as Precedence Constraint Posting (PCP [6]), because it revolves around imposing precedence constraints (the *solution constraints*) on the TCN in order to solve the resource conflicts, rather than fixing rigid values to the start times.

In [4] it is shown that though the previous schedule representation inherently provides a certain level of flexibility at execution time, it guarantees both a time and resource consistent solution only if specific values from the feasibility intervals are chosen for the time points, as described in the following definition:

Definition 1 (Flexible schedule) *A flexible schedule for a problem $\mathcal{P}$ is a network of activities such that a feasible solution for the problem is obtained by allocating each activity at the temporal lower bound allowed by the network.*

For our experiments, flexible schedules are produced using ISES, an effective procedure for RCPSP/max problems [5].

In order to overcome the limitation imposed by the flexible schedule, i.e. having only one consistent solution, a generalization of the TCN produced by a PCP phase is proposed in works such as [4, 11], in which methods for defining a set of both time and resource feasible solutions are presented. This new representation is called *Partial Order Schedule*:

Definition 2 (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.*

A $\mathcal{POS}$ is a special case of a flexible solution and it can be obtained by replacing the solution constraints with a new set of constraints that impose a stronger condition on the TCN (*chaining constraints*). It should be noted that the importance of choosing the RCPSP/max stems from the need to perform a fair comparison between flexible schedules and partial order schedules. In fact, in the absence of maximum time lags, a $\mathcal{POS}$ always represents an infinite and "complete" set of solutions, as it always allows to avoid a re-scheduling phase (propagating the changes that have occurred is sufficient). In the case of flexible schedules instead, propagation alone is not generally sufficient, as any unexpected change might introduce resource conflicts.

Reactive strategies: Local vs. Global rescheduling. When an unexpected temporal event is acknowledged, the on-line solver (Fig. 2) is in charge of firing a re-scheduling algorithm, in order to produce an "updated solution" that will be used to continue the execution.

As already stated, in the current research scenario it is possible to distinguish between the local and global reactive scheduling approaches, the two differing by the extension of the revision action's effects on the initial schedule. In our schema, we model the local and global approach to rescheduling by enabling different constraint removal strategies on the current solution.More specifically:

[1] RCPSP/max is recognized as a quite complex problem; in fact, even the feasibility version of the problem is NP-hard. The reason for the NP-hardness lies in the presence of maximum time-lags, which inevitably imply the satisfaction of deadline constraints.

– *No-Retraction strategy*: in a local perspective, before each revision, none of the constraints imposed is removed, and the subsequent solution is computed by adding further solution constraints.

– *Retraction strategy*: in a global perspective, before each revision, all the previously imposed *solution constraints* are removed; the subsequent solution is computed by adding new constraints to this "clean" representation.

It is worth remarking that exceptions are made for the constraints which model the dynamic aspects of the progressing execution: these constraints are always preserved.

Modeling the real-world uncertainty. In the present study, we focus our attention on the temporal changes which normally characterize the physical environments. In particular, we produce a set of exogenous events for every simulated schedule execution. For obvious reasons, each set is computed on the basis of the schedule's initial characteristics. For the present analysis, we limit ourselves to the generation of temporal events, such as delays of the activities start times and/or modifications of activity processing times:

– *delay of the activity start time*: activity a_i undergoes a delay of Δ_{st} time units, at $t = t_{aware}$ ($e_{delay} = \langle a_i, \Delta_{st}, t_{aware} \rangle$);

– *change of activity processing time*: activity a_i's processing time p_i is extended by Δ_p time units, at t_{aware} ($e_p = \langle a_i, \Delta_p, t_{aware} \rangle$).

For reasons of space we cannot give a complete account on the event generation here. The reader should refer to [12] for further details.

3 Tweaking Proactiveness and Reactiveness

Algorithm 1 illustrates a particular instantiation of our integrated framework, realized with the scheduling technology described in the previous section. The partition between the off-line and on-line scheduling phase is immediately visible. Any scheduling algorithm, either proactive and reactive, can be used within this framework. The system implements an "all-round" schedule management architecture, which allows to follow the schedule's behavior along its complete lifespan. Moreover, it enables the researcher to combine different strategies and to immediately assess the pros and cons of each chosen option.

In order to distinguish among all the different execution combinations, the algorithm is driven by two flags:

– the flag `pos` allows to distinguish the case in which a $\mathcal{POS}$ is created (case POS), from the case in which a flexible schedule is used (case FS);

Algorithm 1: Solve a scheduling problem and Execute one of its solution

```
Input: problem P, policies parameter retract and pos
Output: Execution report
// off-line phase
S ← offlineScheduler(P)
if S does not exist then
    STOP (SOLVER FAILURE)
if pos then
    S ← createPOS(S)
// on-line phase
while a disturb E exists do
    if retract then
        if propagation(E, S) fails ∨ S is not resource consistent then
            S ← removeChoice (S)
            if propagation(E,S) fails then
                STOP (EXECUTION FAILURE)
            S ← onlineScheduler(S)
            if S does not exist then
                STOP (EXECUTION FAILURE)
            if pos then
                S ← createPOS(S)
    else
        if propagation(E,S) fails then
            STOP (EXECUTION FAILURE)
        if S is not resource consistent then
            S ← onlineScheduler(S)
            if S does not exist then
                STOP (EXECUTION FAILURE)
```

– the flag `retract` allows to distinguish between the Retraction strategy (case R from "Retract") and the No-Retraction strategy case (NR).

From this distinction we globally recognize the four strategies that will be compared in the experimental analysis, and denoted as: FS-NR, FS-R, POS-NR and POS-R. In particular, to produce flexible schedules we have used the ISES algorithm [5], because of its efficiency on solving RCPSP/max problems. $\mathcal{POS}$s creation (`createPOS`) is instead performed by applying the CHAINING procedure introduced in [11] to the flexible schedules previously obtained with ISES.

For the on-line phase, we slightly modified the previous procedures to obtain computationally lighter versions. It should be reminded here that the original procedures, being both based on an iterative schema, are CPU expensive: the modified versions are therefore implemented so as to stop the computation as soon as a first viable solution is found, in order to reduce the computational effort. As we will observe in the following section, the on-line versions of the algorithms will tend to spoil the initial solution quality.

Finally, in order to further clarify the algorithm, two more points should be remarked: (a) regardless the type of schedule considered (flexible or partial order), the on-line phase of the algorithm is always initiated with the earliest start time solution; (b) the `propagation()` function

represents a call to the temporal propagation on the TCN.

The Experiment. In this set of experiments we perform the assessment on off-line and on-line phase combinations, by measuring the execution performances in terms of number of successful executions, number of necessary reschedulings, average differences in the final makespan, etc. Moreover, we measure the differences in CPU time between the off-line solving and the rescheduling phase, depending on the various policies chosen.

The comparison presented in this section is based on the analysis of eight different combinations, each obtained coupling two scheduling problem benchmarks, $j30$ and $j100$ [10], with four reactive scheduling benchmarks. The former consist of two sets of respectively 270 and 540 scheduling problem instances of different size, namely 30×5 and 100×5 (number of activities $\times$ number of resources). On the other side, each instance of the reactive scheduling benchmark is composed of a set of properly modeled disturbing events (each set representing a "world simulation"). Each problem belonging to the scheduling benchmark[2] is executed with four instances of world simulations of different size (1, 2, 3, and 5 events each).[3] As stated above, each event represents either a delay on the start time, or a delay on the end time of the activities. These two different kinds of event are produced with the same probability.

4 Results Analysis

Tables 1 and 2 show the results of our investigation for the FS-R, POS-R, FS-NR, and POS-NR execution strategy. To make the comparison more complete, we add a further execution mode based on the use of fixed time solutions, where each activity is assigned a single start time instead of a set of alternatives.

Table 1 contains important information about the number of successfully terminated executions (*executed*) for every strategy, as well as the reason for failure: the *failed resch.* column counts the cases where the execution fails because the on-line solver does not succeed in finding an alternative solution; the *refused events* column counts the cases where the execution fails because the exogenous events are refused due to a constraint violation in the TCN (see section 2). Column *#d* partitions the table in terms of number of disturbs injected in every execution.

[2]Not all the problems belonging to the scheduling benchmark admit a solution; moreover, ISES does not perform a systematic search. For these reasons, our experimental analysis will necessarily be based on the subset of initially solved problems, which amounts respectively to 184 and 480 instances for $j30$ and $j100$.

[3]The execution of a project scheduling problem can not be considered completed until all the activities are successfully processed: this implies that any variation to the nominal conditions which may occur at execution time is likely to require the synthesis of a new schedule. This represents a major difference with respect to other dynamic problems where the execution of certain activities can be overruled.

method	#d	j30			j100		
		executed	failed resch.	refused events	executed	failed resch.	refused events
FS-R	**1**	91,85%	1,08%	7,07%	91,04%	2,08%	6,88%
POS-R		89,13%	0,54%	10,32%	87,29%	2,08%	10,63%
FS-NR		91,84%	1,09%	7,06%	91,87%	1,25%	6,87%
POS-NR		88,04%	1,63%	10,32%	86,87%	2,50%	10,62%
fixed time		90,76%	5,43%	3,80%	89,79%	3,75%	6,45%
FS-R	**2**	87,50%	4,90%	7,60%	85,21%	3,13%	10,66%
POS-R		80,43%	3,27%	16,30%	76,46%	2,29%	21,25%
FS-NR		88,58%	3,26%	8,15%	85,62%	2,71%	11,66%
POS-NR		80,97%	3,26%	15,76%	73,95%	5,00%	21,04%
fixed time		82,61%	13,04%	4,34%	81,25%	8,54%	10,20%
FS-R	**3**	85,33%	7,06%	7,61%	79,17%	3,96%	16,87%
POS-R		76,63%	3,26%	20,11%	69,58%	2,92%	27,50%
FS-NR		83,15%	5,43%	11,41%	80,00%	2,50%	17,50%
POS-NR		75,00%	3,80%	21,20%	67,71%	5,41%	26,87%
fixed time		78,26%	16,30%	5,43%	77,50%	6,87%	15,62%
FS-R	**5**	75,00%	6,52%	18,48%	70,21%	4,17%	25,63%
POS-R		57,61%	2,72%	39,67%	60,42%	3,13%	36,46%
FS-NR		74,45%	4,34%	21,20%	70,41%	3,33%	26,25%
POS-NR		58,15%	2,71%	39,13%	56,66%	7,08%	36,25%
fixed time		76,63%	13,58%	9,78%	67,08%	8,33%	24,58%

Table 1: Success rate of each execution strategy

The most immediate (and somehow surprising) effect that the dynamic analysis reveals is the POS-R/POS-NR significantly lower success rate in terms of completed executions, with respect to all other methods. As shown in the table, the reason lies in a dramatic increase in the number of rejected disturbs (*refused events* column). This apparent anomaly can be explained in terms of TCN "constrainedness": in fact, the creation of a $\mathcal{POS}$ inherently requires a higher number of temporal constraints with respect to a flexible schedule, in order to guarantee a resource conflict-free solution. This inevitably makes the TCN more reluctant to accept new disturbing events during the execution phase; this circumstance is supported by the worsening of the same effect as the number of events increases.

As readily observable, the case which exhibits the highest rate of unsuccessful reschedulings is the execution of *fixed time* solutions. This is due to the fact that a fixed time solution requires a rescheduling for *every* disturbing event, highly stressing the on-line scheduler (which in our case performs an incomplete search), and therefore increasing the possibility of failure. Again, it can be seen that the more events are injected, the more evident the effect.

Table 2 presents a different set of results: for each execution policy, it shows the average makespan at the end of every execution (*mk*), the average difference between the makespan at the beginning and at the end of every execution (Δmk), the number of performed reschedulings (*reschedulings*) related to the number of injected disturbs, the average CPU time necessary to compute the initial schedule (*CPU off-line*), and finally, the average CPU time which is necessary to perform all the reschedulings during the execution (all CPU times are expressed in milliseconds). For a fair comparison of the different policies, all the averages presented in this table are computed on the basis of the problem instances commonly executed with

method	#d	j30					j100				
		mk	Δmk	resche-dulings	CPU off-line	CPU on-line	mk	Δmk	resche-dulings	CPU off-line	CPU on-line
FS-R	**1**	103.43	4.29	27.27%	4478.31	77.34	424.60	9.02	24.38%	52599.11	766.15
POS-R		102.63	3.60	5.19%	4613.57	15.13	419.88	5.07	11.58%	36287.57	303.97
FS-NR		102.56	3.43	27.27%	4481.10	14.68	419.06	3.48	24.14%	36242.48	130.74
POS-NR		102.14	3.10	5.19%	4612.60	2.34	417.11	2.31	11.58%	36251.86	54.59
fixed time		106.29	7.16	100.00%	4480.00	300.45	437.36	21.78	99.75%	67538.68	3035.68
FS-R	**2**	106.99	8.02	34.21%	4106.09	150.53	435.54	13.95	23.04%	30347.49	874.70
POS-R		104.91	6.03	4.51%	4242.03	20.23	429.22	8.41	10.03%	38529.15	674.86
FS-NR		104.90	5.93	34.21%	4104.89	34.51	427.90	6.30	22.88%	30271.00	258.50
POS-NR		104.83	5.95	4.51%	4239.40	3.83	424.74	3.93	9.56%	32371.82	97.46
fixed time		107.35	8.38	100.00%	4108.80	500.98	446.62	25.02	99.53%	30259.15	2768.37
FS-R	**3**	109.55	9.73	26.90%	4506.40	190.00	449.84	19.40	20.27%	26393.55	963.16
POS-R		108.11	8.36	5.85%	4647.54	37.02	441.49	11.93	9.41%	33716.81	675.18
FS-NR		108.00	8.18	26.61%	4515.44	39.91	439.66	9.23	22.37%	26318.37	371.03
POS-NR		107.83	8.09	5.85%	4646.23	7.72	436.24	6.68	9.63%	28406.98	151.26
fixed time		109.95	10.12	100.00%	4511.93	848.42	458.32	27.89	99.22%	26300.50	3716.15
FS-R	**5**	119.30	16.19	26.43%	4281.90	267.38	464.12	28.85	22.45%	25792.36	2391.92
POS-R		116.44	13.37	5.24%	4413.81	73.33	455.56	21.07	10.57%	33162.71	1748.56
FS-NR		117.12	14.01	22.86%	4277.38	59.52	447.42	12.15	21.66%	25682.40	646.90
POS-NR		115.86	12.79	5.00%	4410.60	12.14	444.68	10.18	10.48%	27544.98	289.17
fixed time		118.33	15.23	100.00%	4286.19	1161.19	465.56	30.29	98.43%	25754.32	6721.48

Table 2: Summarizing data for each execution strategy (computed on the intersection set of all successfully executed problems)

all the execution strategies.

One of the most important characteristic to be observed is the extremely low rate of necessary reschedulings exhibited by the POS-R/POS-NR policies: this result is all but surprising and confirms the theoretical expectations which motivated the study on the $\mathcal{POS}$. As shown, the need for schedule revision in case of $\mathcal{POS}$ utilization roughly decreases by more than 75% in case of the j30 set, and by at least 50% in case of the j100 set with 5 disturbs. Note also the $\approx 100\%$ *reschedulings* figure relative to the case of *fixed time* schedules: in this case, a schedule revision is practically always needed: this is confirmed by the extremely high *CPU on-line* values, especially for the j100 set. Moreover, the fixed time strategy reveals the highest rates of makespan elongation (see *mk* and Δ*mk*): in fact, the frequent rescheduling actions, being performed by a less specialized makespan-optimizing procedure to speed up reaction (`onlineScheduler` in Algorithm 1), inevitably tend to spoil makespan quality.

A delicate question should be raised at this point, in order to acquire a better understanding of the presented results: in many scheduling contexts, solution *continuity* is a fundamental issue. In our experiments we try to maintain schedule continuity through the NR technique, by guaranteeing that every new solution 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 reshuffling 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 R strategies (which allow a greater re-shuffling) to return better makespan values with respect to NR strategies. Indeed, this is exactly what would happen if we decided to give up reaction times and utilize the same algorithm for both off-line and on-line scheduling (as other experiments confirm); but in the present analysis, such expectations are frustrated because the on-line scheduling algorithm severely spoils the makespan of the initial schedule.

Another interesting aspect is related to the comparison of the *CPU on-line* values between the *Retraction* and *No-Retraction* strategies: in fact, it should be noticed that NR strategies reveal lower CPU-load rates with respect to the R strategies, *despite the comparable amount of performed reschedulings*. As explained in section 2, NR execution modes retain all the temporal constraints of the previous solution: hence, the rescheduler is bound to work on a smaller search space, finding the next solution almost immediately. Additionally, it should be noticed the tremendous on-line CPU load in the case of fixed time solutions, as well as how the reactiveness of NR technique, combined with the low revision requirements featured by the $\mathcal{POS}$, allows for the fastest dynamic reactions of the whole set (see the *CPU on-line* values for POS-NR).

5 Conclusions and Future Work

Our work is based on the assumption that scheduling is a process where the proactive and reactive phases represent a *continuum*. In particular this paper introduces a schedule management schema where the off-line and on-line approaches are not mutually exclusive. This schema allows to have a more informed evaluation focusing the attention on the combination of off-line/on-line scheduling techniques.

As shown in the preceding sections, several interesting features are in fact emphasized by this integrated evaluation of proactive and reactive phases. Some results confirm the expectations while others require a certain level of analysis in order to be correctly understood. For instance, the rigid behavior exhibited by the fixed time schedules when confronted with dynamically variable environments is totally confirmed, as confirmed is the behavior of schedules characterized by a more flexible nature.

However, the scarce capability in accepting exogenous events which afflicts the $\mathcal{POS}$ was not so predictable, and would not have been revealed without a proper testing. The experiments shed light on many possible trade-offs that must be faced, depending on the conditions of execution as well as on the particular aspects that must be privileged. If fast responsiveness is a primary concern, the $\mathcal{POS}$ is extremely efficient because it rarely needs rescheduling, as opposed to flexible schedules. The price to pay for this ability is a higher chance that the event is not accepted by the TCN underlying the problem. Moreover, if solution stability is important, the experiments show that the $\mathcal{POS}$ is to be preferred, as it naturally maintains a higher level of continuity.

The results of this integrated evaluation suggest several research lines which are the object of ongoing work, such as the production of a different class of $\mathcal{POS}$s, through the development of alternative chaining procedures aimed at minimizing the inevitable increase of constrainedness in the TCN, followed by the same on-line experimentation; the latter being necessary in order to highlight possible undesired side-effects.

As another example, the observed dynamic behavior of the schedules suggests to study the introduction of different reactive techniques. For instance, a possible approach may be based on *informed retraction* procedures, where the constraint removal strategy is preceded by a search phase to determine the constraints which are to be retracted, depending on the particular dynamic requirements.

Acknowledgments. Amedeo Cesta, Nicola Policella and Riccardo Rasconi's work is partially supported by MIUR (Italian Ministry for Education, University and Research) under project ROBOCARE (L. 449/97) and contract #2005-015491 (PRIN).

References

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

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

[3] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research*, 112:3–41, 1999.

[4] A. Cesta, A. Oddi, and S. F. Smith. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the 4^{th} International Conference on Artificial Intelligence Planning Systems, AIPS-98*, pages 214–223. AAAI Press, 1998.

[5] A. Cesta, A. Oddi, and S. F. Smith. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the 16^{th} International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann, 1999.

[6] C. Cheng and S. F. Smith. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings of the 12^{th} National Conference on Artificial Intelligence, AAAI-94*, pages 1086–1091. AAAI Press, 1994.

[7] A. J. Davenport and J. C. Beck. A Survey of Techniques for Scheduling with Uncertainty. accesible on-line at http://tidel.mie.utoronto.ca/publications.php on June 2006, 2000.

[8] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

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

[10] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark Instances for Project Scheduling Problems. In J. Weglarz, editor, *Project Scheduling - Recent Models, Algorithms and Applications*, pages 197–212. Kluwer Academic Publishers, Boston, 1998.

[11] N. Policella, A. Oddi, S. F. Smith, and A. Cesta. Generating Robust Partial Order Schedules. In *Principles and Practice of Constraint Programming, 10^{th} International Conference, CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 496–511. Springer, 2004.

[12] N. Policella and R. Rasconi. Designing a Testset Generator for Reactive Scheduling. *Intelligenza Artificiale*, 3:29–36, 2005.

[13] R. Rasconi, N. Policella, and A. Cesta. SEaM: Analyzing Schedule Executability through Simulation. In *The 19^{th} International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE'06)*, 2006.

[14] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments – a survey. *Constraints*, 10(3):253–281, 2005.