SEaM: Analyzing Schedule Executability Through Simulation

Riccardo Rasconi*, Nicola Policella, and Amedeo Cesta

ISTC-cnr
Institute for Cognitive Science and Technology
National Research Council of Italy
name.surname@istc.cnr.it

Abstract. Increasing attention is being dedicated to the problem of schedule execution management. This has encouraged research focused on the analysis of strategies for the execution of plans in real working environments. To this aim, much work has been recently done to devise scheduling procedures which increase the level of robustness of the produced solutions. Yet, these results represent only the first step in this direction: in order to improve confidence in the theoretical results, it is also necessary to conceive experimental frameworks where the devised measures may find confirmation through empirical testing. This approach also has the advantage of unveiling possible counter-intuitive insights of the proposed scheduling strategies, which otherwise might remain concealed. This paper presents: (a) an experimental platform designed to tackle the problem of schedule execution with uncertainty; (b) an analysis of a variety of schedule execution tests performed under variable environmental conditions.

1 Introduction

One of the most relevant issues in Scheduling 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. The growing attention dedicated to this specific issue in research areas such as OR and AI is proved by the increasing number of single results and surveys [1, 2, 3, 4]. Notwithstanding the relatively recent developments, the whole topic still offers much room for investigation.

The aim of our work is to compare different approaches to schedule execution under uncertainty in a fair and controlled way, specifically focusing on Project Scheduling Problems [5]. These problems are characterized by a rich internal structure. They are based on a network of activities, among which it is possible to identify complex temporal relations that can be used to model a number of variably rigid causal links which normally constrain the tasks in a project. As a further source of complexity, several heterogeneous resources with different capacities serve the activities according to complex modalities.

This paper presents an experimental framework which allows us to carry out a series of reproducible experiments which aim at examining the behavior of schedules under

* PhD Student at DIST - University of Genova.

M. Ali and R. Dapoigny (Eds.): IEA/AIE 2006, LNAI 4031, pp. 410–420, 2006.

execution, upon the occurrence of different kinds of unexpected disturbances. To show the possibilities opened by the framework we present a set of comparison on different state-of-the-art constraint based techniques. The experiments we present yield different results depending on a number of crucial factors, which range from the particular technique used to synthesize the initial solution (baseline schedule), to the type of rescheduling methodology employed for solution revision.

The paper is organized as follows: Section 2 briefly introduces the schedule execution problem we tackle, as well as the reference scheduling problem the whole framework is based upon; Section 3 describes the whole experimental infrastructure, giving special attention to the types of flexible solutions we use as baseline schedules and to the execution algorithm used to simulate their execution; Section 4 briefly introduces how uncertainty is modeled in the framework, while Section 5 describes the experiments and the obtained results. Section 6 concludes the paper.

2 The Scheduling Problem Under Uncertainty

In general terms, solving a scheduling problem for a given set of activities (or tasks) that have to be executed, basically means to find a suitable temporal allocation for each activity, so as to guarantee "good" performance relative to the optimization of an objective function, usually the project total completion time (*makespan*).

What follows is a brief description of the particular problem that we will refer to throughout all the paper, known as *Resource-Constrained Project Scheduling Problem with minimum and maximum time lags* (RCPSP/max) [6], which is basically 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.

This class of problems is interesting because it allows to model a broad range of real domains, that is, domains in which complex causal relations between activities and coordination between multiple steps must be enforced. These problems are characterized by a rich variety of time and resource constraints. The price for such expressiveness is that RCPSP/max is a complex problem; in fact, both optimization and feasibility are NP-hard[1].

[1] The reason for the NP-hardness lies in the presence of maximum time-lags, which inevitably imply the satisfaction of deadline constraints, thus transforming feasibility problems for precedence-constrained scheduling into scheduling problems with time windows.

Clearly, our goal is not limited to solving the RCPSP/max: in fact, regardless of the complexity related to finding a solution of a scheduling problem, other kinds of difficulties arise when we try to execute it in real working environments. The unpredictability which affects real world domains inherently entails a high level of uncertainty on the execution conditions. Therefore, unless some intelligent actions are taken during either the initial solving process and/or the execution phase, the solution is bound to lose its consistency, and therefore its usefulness.

Execution uncertainty in scheduling can be dealt with either *proactively* or *reactively*. The key point in the proactive approach is to synthesize solutions that are able to absorb the effects of unexpected events, thus minimizing the need to reschedule, while the reactive approach requires to exploit local or global online adjustments to the schedule, in order to re-gain consistency. As we will see, the difference between these two strategies is often very subtle; the objective of this paper is to investigate the possibilities offered by trading-off among simple adjustments, fixes to the baseline schedule, and new calls to problem solvers. To this aim, we have built an experimental environment targeted at studying how increasingly robust baselines can influence the reactive phase. Clearly, acting proactively requires a greater effort during the initial solving process in order to grant robustness to the solution, while acting reactively generally makes the execution phase more demanding, because of the higher number of necessary reschedulings. In order to make the reactive approach computationally lighter, it is possible to bind the scope of the reaction within an arbitrarily wide neighborhood of the schedule conflicting area (local reaction). The global approach in general guarantees to find a higher quality solution with respect to a local method, though it exhibits a lack in reactivity, requiring more time for system reconfiguration. This paper addresses the issue of local vs. global re-scheduling within a particular set of constraint-based algorithms which are global in nature, but are forced to act locally by over-constraining the updated problem that the scheduler is called to reason upon.

It is worth noting that the problem of execution has been addressed in AI planning by several works [7, 8, 9], nevertheless those solutions are not directly applicable to project scheduling and differs from those described in this paper.

3 *SEaM:* The Scheduling Execution and Monitoring Framework

Fig. 1(a) shows the complete platform that we have set up in order to produce the experiments. It is composed of three blocks: the *solver* and the *generator* work off-line and have the job of, respectively, computing the initial solution (the baseline schedule), and generating the exogenous events, intended to disturb the schedule during its execution; the third block, called *SEaM* (for *Schedule Execution and Monitoring*), works on-line, and is responsible for performing a complete simulation of the execution of the initial solution. The disturbing events synthesized by the *generator* are injected during the simulated execution at specified times, and their effects are counteracted by the *SEaM* module, which is in charge of maintaining schedule consistency by exploiting a number of different *re-scheduling* policies.

The core of our work is represented by *SEaM* which is depicted in more detail in Fig. 1(b). This component accepts in input: (1) the initial solution (i.e. the schedule that

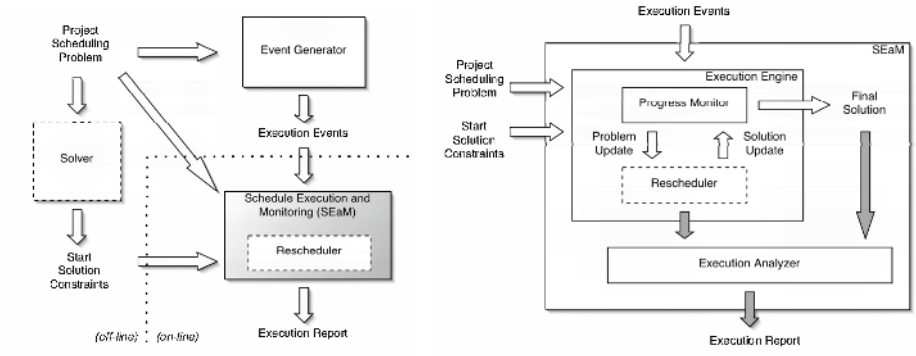


Fig. 1. The platform for comparing execution strategies (a), and detail of the *SEaM*module (b)

will be executed), in terms of the initial scheduling problem plus the constraints produced in the solving process, and (2) the exogenous events which have been produced by the event generator. Subsequently, the *Progress Monitor* module manages the schedule execution by simulating the advancement of time and dispatching the activities at their earliest possible start time, so as to minimize the global makespan. When an unexpected temporal event is acknowledged, the current problem is updated and passed to the *Rescheduler* which is in charge of producing the new solution that will act as the next baseline to continue execution. The cycle proceeds until either the execution is succesfully completed or a failure in delivering a new solution in encountered: in the latter case, the simulation is aborted. The figure shows also an additional module, the "Execution Analyzer", that is responsible for monitoring the whole execution process and finally generates the reports that describe the outcome of the experiments.

3.1 Solution Representation: Flexible Schedules vs. $\mathcal{POS}$s

Going back to Fig. 1(a), we can observe that the module in charge of enacting the strategy in order to deal with uncertainty is the off-line solver, which produces the baseline schedule. In contrast to most off-line schedulers, which deliver fixed-time solutions where the decision variables are represented by the start times of each activity, our approach is based on the general concept of "temporal flexibility" [10]. 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 second network (Temporal Constraint Network – TCN [11]), 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. The search schema used in our approach focuses 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 the Precedence Constraint Posting (PCP) [10], because it

revolves around imposing precedence constraints (the *solution constraints*) on the TCN to solve the resource conflicts, rather than fixing rigid values to the start times.

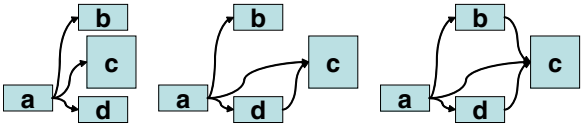
In [12] it is shown that though the previous schedule representation inherently provides a certain level of resilience at execution time, it guarantees both a time and resource consistent solution only if specific values from the feasibility intervals are chosen for the time points, as described in the following definition:

Definition 1 (Flexible schedule). *A flexible schedule for a problem $\mathcal{P}$ is a network of activities, (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.*

For our experiments, flexible schedules are produced through the ISES procedure described in [13]. This algorithm has proved to be effective on RCPSP/max problems. In order to overcome the limitation imposed by the flexible schedule of having only one consistent solution, a generalization of the TCN produced by a PCP phase is proposed in works such as [12, 14], in which methods for defining a set of both time and resource feasible solutions are presented. This new representation is called *Partial Order Schedule* [14]:

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*). Fig. 2 shows an example of different types of scheduling solutions: the problem (a) is composed of four activities *a*, *b*, *c*, and *d*, all using the same resource, (maximum capacity = 2). Solution (b) represents a flexible schedule where the resource conflict has been solved by adding the solution constraint $d \prec c$; solution (c) represents a $\mathcal{POS}$ where two chaining constraints have been added: $b \prec c$ and $d \prec c$. 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.



(a) Problem (b) Flexible schedule (c) $\mathcal{POS}$

Fig. 2. A problem and different solution representations

3.2 The Execution Algorithm

The details of the execution algorithm are illustrated in Algorithm 1. The algorithm is divided in an off-line and an on-line section; in the former, the initial solution can be

Algorithm 1. Solve a scheduling problem P and Execute one of its solution S

```
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)
            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)
```

computed by the `offlineScheduler()` either as a flexible schedule (case FS) or as a $\mathcal{POS}$ (case POS) through the `createPOS()` procedure, depending on the value of the flag `pos`. In the latter, and regardless how the initial solution is produced, it is put into execution according to different modalities, depending on the value of the flag `retract`. At each step of the execution cycle, the environment is sensed for possible disturbs. Afterwards, if `retract` $= true$, the execution algorithm firstly removes all the constraints imposed in the previous solving process (`removeChoice()`), and secondly looks for a new solution (`onlineScheduler()`), possibly creating a new $\mathcal{POS}$. Note that the `onlineScheduler()` procedure implements the *Rescheduler* dotted box in Figs. 1(a) and 1(b). If `retract` $= false$, a new solution is searched leaving the previously imposed solution constraints untouched. In both cases, the algorithm initially checks for temporal consistency after each disturb is acknowledged through the `propagation()` procedure. Temporal consistency is in fact a condition that must be satisfied at all times, in order to continue schedule execution. The simulation is carried on until all the activities of the schedule are successfully executed, a temporal inconsistency is detected or the rescheduler fails to find an alternative solution.

4 Modeling the Uncertainty

The elements of uncertainty which usually impair the consistency of a schedule, may belong to one of the following types: (1) temporal changes, which exclusively pertain the temporal aspects of the problem; (2) resource variations, which modify the resource availability during the execution of a schedule; (3) causal changes, which involve the

introduction of new constraints among the activities; (4) variations in the number of activities that have to be executed. In the present study, we focus our attention on the temporal aspects connected to the unpredictability of real environments: in particular, we will limit ourselves to the generation of temporal events, such as sudden delays of the activity start times and/or unexpected lengthening of the activity processing times, as depicted in Fig. 3:

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

change of activity processing time: $e_p = \langle a_i, \Delta_p, t_{aware} \rangle$
activity a_i's processing time p_i is extended by Δ_p time units, at $t = t_{aware}$.

It is outside the scope of this paper to give a complete account on the event generation: the interested reader should refer to [15] for further details. Yet, the aspect worth highlighting here is the rationale we have pursued in studying the generation of unexpected events. In fact, though in the present work we produce the disturbances on top of the scheduling benchmark set described in [16], it should be remarked that the event-generating procedure we have devised is quite general, and can be used on a variety of different reference problems.

In the production of a benchmark set, we identified a number of fundamental features which characterize each unexpected event, such as the event type, the event quality (or magnitude), as well as the its temporal distance from the previous event. The spacing of different exogenous events in each instance has been realized by adding the parameter t_{aware} in the definition of each event. This element specifies the "absolute" instant where the specific event is supposed to occur. By using the t_{aware} parameter it is possible to temporally sort all the generated events and to "fire" them in order of occurrence.

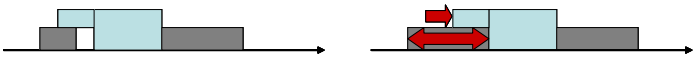


Fig. 3. Temporal changes: activities can last more than expected or they can undergo delays

5 Experimental Results

The comparison presented in this section is based on the analysis of known standard scheduling benchmarks in RCPSP/max combining each of them with four reactive scheduling benchmarks (world simulations). In this paper we present the results obtained from the $j30$ set [16]. This particular problem benchmark consists of one set of 270 scheduling problem instances of size 30×5 (number of activities $\times$ number of resources). Each instance of the reactive scheduling benchmark is composed of a set of unexpected events: each problem in the scheduling benchmark is associated with four instances of world simulations of different size (1, 2, 3, and 5 events each)[2]. In other

[2] All events represent either a delay on the start time, or a delay on the end time of the activities, and they are produced with the same probability throughout all the world simulations.

words, every scheduling problem will be put into four simulated executions, where each execution will be disturbed by, respectively, 1, 2, 3 and 5 exogenous events.

Table 1 shows the results of the executions on the j30 scheduling benchmark. We evaluate the different combinations of off-line/on-line policies — FS-NR (Flexible Schedule with No-Retraction), FS-R (Flexible Schedule with Retraction), POS-NR (POS with No-Retraction) and POS-R (POS with Retraction). In order 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. For each column in the table, we take into account the following aspects: the number of disturbances injected during each single execution (*number of disturbs*), the percentage (with respect to the number of initially solved problems) of the schedules which successfully completed the execution (*% executed*), the average makespan of the successfully executed schedules (*mk*), the average makespan deviation of the schedules before and after the execution (Δ *mk*), the percentage of the performed re-scheduling actions with respect to the number of the injected disturbances (*% resched.*), the average CPU time, in msecs, to compute the initial solution (*CPU Off-line*), the average CPU time spent to perform all re-schedulings during the execution (*CPU On-line*), the sensitivity of activity start time w.r.t. the execution process (ψ): this measure gives an assessment of how much the initial solution has been affected by the occurrence of the exogenous events during the execution[3].

With the exception of the *% executed* column, all data in the table are computed on the intersection set of all the succesfully executed problems.

5.1 Result Analysis

One of the most striking results that we observe regards the different abilities in preserving the executability of a solution. The outcome shows that the use of partial order schedules tends to lower the success rate in terms of completed executions (*% executed* column), mainly due to the dramatic increase in the number of rejected disturbs during the execution. This apparent anomaly can be explained as follows: the creation of a $\mathcal{POS}$ inherently involves a higher level of "constrainedness" in the TCN, in order to guarantee a resource conflict-free solution. This circumstance inevitably makes the TCN more reluctant in accepting new contraints, in the specific case, the constraints which model the exogenous events. Also, note how this effect gets worse as the number of the exogenous events increases (88,04% in the POS-NR case with 1 event, against 58,15% with 5 events).

Another important characteristic to be observed is the extremely low rate of necessary reschedulings exhibited by the POS-R/POS-NR policies (*% resched.* column): 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 sensibly decreases by more than 75% in case of 5 disturbs. Note also the 100%

[3] This parameter is currently computed as follows $\psi = \sum_{i=1}^{N} \frac{|st_f(a_i) - st_0(a_i)|}{N}$. This measure gives an assessment of how much the initial solution has been affected by the occurrence of the exogenous events during the execution. The lower the value, the more the solution proved to be stable.

reschedulings figure relative to the case of *fixed time* schedules: in this case, a schedule revision is always needed: this is also confirmed by the extremely high *CPU on-line* values.

A very interesting aspect can be observed by comparing the *CPU on-line* values between the *Retraction* and *No Retraction* strategies. In general, the Retraction methods require a higher CPU on-line load because the removal of the solution constraints inevitably re-introduces some resource conflicts that must be solved by rescheduling. But the intriguing result lies in the fact that this difference in the CPU on-line rates stands *despite the comparable amount of performed reschedulings*. Let us look at the difference between the FS-R and FS-NR rates: it can be seen that, in the 5 events case, we have 267 ms. (FS-R) against 59 ms. (FS-NR), although the number of performed reschedulings is very close ($\approx 23\% \div 26\%$)! The same effect can be observed between the POS-R and POS-NR cases: 73 ms. against 12 ms., notwithstanding the same ($\approx 5\%$) number of reschedulings. This circumstance can be explained as follows: 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. The table also shows how inefficient the executions of fixed time solutions are (see the CPU on-line values).

Table 1. j30 Execution Results

method	number of disturbs	% executed	mk	Δmk	% resched.	CPU off-line	CPU on-line	ψ
FS-R		91.85%	103.43	4.29	27.27%	4478	77	2.55
POS-R		89.13%	102.63	3.60	5.19%	4613	15	1.84
FS-NR	**1**	91.85%	102.56	3.43	27.27%	4481	15	1.68
POS-NR		88.04%	102.14	3.10	5.19%	4612	2	1.55
fixed time		90.76%	106.29	7.16	100.00%	4480	300	5.04
FS-R		87.50%	106.99	8.02	34.21%	4106	150	4.32
POS-R		80.43%	104.91	6.03	4.51%	4242	20	2.92
FS-NR	**2**	88.59%	104.90	5.93	34.21%	4105	34	2.87
POS-NR		80.98%	104.83	5.95	4.51%	4239	4	2.83
fixed time		82.61%	107.35	8.38	100.00%	4109	501	5.57
FS-R		85.33%	109.55	9.73	26.90%	4506	190	5.40
POS-R		76.63%	108.11	8.36	5.85%	4647	37	4.30
FS-NR	**3**	83.15%	108.00	8.18	26.61%	4515	40	3.98
POS-NR		75.00%	107.83	8.09	5.85%	4646	8	4.02
fixed time		78.26%	109.95	10.12	100.00%	4512	848	6.45
FS-R		75.00%	119.30	16.19	26.43%	4282	267	8.48
POS-R		57.61%	116.44	13.37	5.24%	4414	73	6.48
FS-NR	**5**	74.46%	117.12	14.01	22.86%	4277	59	6.91
POS-NR		58.15%	115.86	12.79	5.00%	4410	12	6.15
fixed time		76.63%	118.33	15.23	100.00%	4286	1161	8.68

One last word on schedule stability: the *Δmk* and ψ columns give a measure of the ability of a solution to remain stable during the execution. The former column takes into account solution continuity in terms of makespan preservation, while the latter accounts for stability in terms of maintainace of the activities' start times. As it can be seen, the highest stability during execution is obtained with the NR as opposed to R methodologies, because they maintain the structure of the initial solution for the whole execution. The most unstable solutions are produced in the fixed time case, as a direct consequence of the complete lack of temporal flexibility. In the cases where it is essential to maintain a high level of solution continuity, the No-Retraction methods are therefore to be preferred.

6 Conclusions

It is straightforward to assume that in order to face project scheduling problems, it is necessary to consider both the off-line solving phase, as well as a dynamical management, where the behaviour of the produced schedule is followed and analyzed throughout its whole lifespan. This paper discusses alternative approaches to the production, execution, monitoring and repairing of pre-defined schedules enforced in a constraint-based framework. We presented a platform, named *SEaM*, that simulates the execution of a baseline schedule, by maintaining an internal representation of the solution and updating this representation in order to (a) take into account the occurrence of exogenous events and (b) counteracting the effects of such events through proper rescheduling.

The *SEaM* allows us to obtain different insights on the combination of two aspects: (a) the use of different off-line scheduling techniques to increase proactive robustness and (b) the use of complementary re-scheduling policies to react to unexpected events. In particular we analyze two alternatives to the classical fixed time schedule – flexible schedules, containing a single point solution, and partial order schedules or $\mathcal{POS}$s. Moreover, we distinguish between the incremental modification of the initial schedule vs. the retraction of previously made decisions followed by a new resolution.

On the basis of well known project scheduling problem benchmarks, a set of exogenous events are produced *ad hoc* for each problem instance in order to maximize the possibility of dynamical event-acceptance. It is also worth remarking that the produced events are independent from the particular off-line scheduler used to provide the initial solution, as well as from the type of initial solution itself. The previous characteristics make it possible to use the *SEaM* framework to perform reproducible experiments.

A set of results support the usefulness of this analysis. Part of the outcome represents a direct confirmation of theoretical expectations from previous analysis; yet, the presence of counterintuitive insights in the results opens the way for further investigation and new perspectives.

Acknowledgments. The authors' work is partially supported by MIUR (Italian Ministry for Education, University and Research) under projects RoboCare (L. 449/97) and "Vincoli e Preferenze" (PRIN).

References

1. Davenport, A.J., Beck, J.C.: A Survey of Techniques for Scheduling with Uncertainty. accessible on-line at http://tidel.mie.utoronto.ca/publications.php on Feb, 2006. (2000)
2. Aytug, H., Lawley, M.A., McKay, K.N., Mohan, S., Uzsoy, R.M.: Executing production schedules in the face of uncertainties: A review and some future directions. European Journal of Operational Research **165** (2005) 86–110
3. Herroelen, W., Leus, R.: Robust and reactive project scheduling: a review and classification of procedures. International Journal of Production Research **42** (2004) 1599–1620
4. Verfaillie, G., Jussien, N.: Constraint solving in uncertain and dynamic environments – a survey. Constraints **10** (2005) 253–281
5. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. European Journal of Operational Research **112** (1999) 3–41

6. Bartusch, M., Mohring, R.H., Radermacher, F.J.: Scheduling project networks with resource constraints and time windows. Annals of Operations Research **16** (1988) 201–240
7. Schoppers, M.: Universal Plans for Reactive Robots in Unpredictable Environments. In: Proceedings of the Tenth International Joint Conference on Artificial Intelligence, IJCAI-87. (2005)
8. Ambros-Ingerson, J., Steel, S.: Integrating Planning, Execution and Monitoring. In: Proceedings of the Seventh National Conference on Artificial Intelligence, AAAI-88. (1988)
9. Beetz, M., McDermott, D.: Improving Robot Plans During Their Execution. In: Proceedings of the Second International Conference on AI Planning Systems, AIPS-94. (1994)
10. Cheng, C., Smith, S.F.: Generating Feasible Schedules under Complex Metric Constraints. In: Proceedings of the 12^{th} National Conference on Artificial Intelligence, AAAI-94, AAAI Press (1994) 1086–1091
11. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991) 61–95
12. Cesta, A., Oddi, A., Smith, S.F.: 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, AAAI Press (1998) 214–223
13. Cesta, A., Oddi, A., Smith, S.F.: An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In: Proceedings of the 16^{th} International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1999) 1022–1029
14. Policella, N., Oddi, A., Smith, S.F., Cesta, A.: 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., Springer (2004) 496–511
15. Policella, N., Rasconi, R.: Designing a Testset Generator for Reactive Scheduling. Intelligenza Artificiale (Italian Journal of Artificial Intelligence) **II** (2005) 29–36
16. Kolisch, R., Schwindt, C., Sprecher, A.: Benchmark Instances for Project Scheduling Problems. In Weglarz, J., ed.: Project Scheduling - Recent Models, Algorithms and Applications. Kluwer Academic Publishers, Boston (1998) 197–212