Testsets Generation for Reactive Scheduling

N. Policella and R. Rasconi
ISTC-CNR
Institute for Cognitive Science and Technology
National Research Council of Italy
{name.surname}@istc.cnr.it

SOMMARIO/*ABSTRACT*

This paper presents some basic ideas for the creation of a benchmark generator for reactive scheduling problem instances. The main motivations behind this work grow out either from the recognized lack (hence the necessity) of benchmark sets for this specific problem as well as from the conviction that the resolution of a scheduling problem consists both in the synthesis of an initial solution ("static" scheduling) and in the utilization of a number of methodologies dedicated to the continuous preservation of solution consistency. In fact, the occurrence of exogenous events during the execution phase in real working environments, often compromises the schedule's original qualities.

1 Introduction

Scheduling is defined as the assignment of start and end times to a set of activities (or tasks) subject to a number of constraints. Task synchronization in the production chain of a factory, management of space missions, and transportation scheduling to support crisis management, are only but a few representative examples. However, the synthesis of initially feasible schedules is hardly ever sufficient as, in real-world working environments, unforeseen events tend to quickly invalidate the schedules predictive assumptions and bring into question the continuing validity of the schedule's prescribed actions. Therefore, the lifetime of a schedule tends to be very short. In general, approaching a scheduling problem requires the coupling of a "predictive" scheduling engine, able to propose a possible solution in a compact representation, and a "reactive" scheduling engine, able to manage the current solution and to make event-triggered decisions at execution time.

Even though the predictive scheduling aspects have been thoroughly evaluated through the production of several benchmark data sets and metrics, the aspect related to reactive scheduling has not yet received the same level of attention. It is our opinion that this trend should be inverted as research on the performance of the reactive scheduling engine is fundamental not only to assess reaction performances, but also to have an evaluation on the suitability and appropriateness of the scheduling approach in its entirety, that is, from the proactive and the reactive point of view.

The paper is organized as it follows: we start recalling a specific family of scheduling problems: the project scheduling problems. Next we describe the issue of schedule execution and the main elements of a testsets for this problem. Section 4 introduces a first benchmark generator for a specific problem: RCPSP/max. Finally, we conclude providing final remarks on future refinements and extensions of this work.

2 Project Scheduling Problems

The scheduling problem is primarily concerned with figuring out *when* tasks should be executed so that the final solution guarantees "good" performance relatively to the optimization of given objective functions. Real-life scheduling problems, like those in industrial applications, typically involve constraints that are often wide ranging and complex in nature. In manufacturing production environments for example, resource allocation decisions must be consistent with capacity limitations, machine setup requirements, etc. Similarly, production activities have associated durations and precedence constraints, and may require the availability of multiple resources (e.g., machines, operators, tooling, raw materials). For these reasons, scheduling problems are generally very complex and it can be shown that many of them are NP-hard. For the purpose of this paper, we focus on a particular family of scheduling problems, known as Project Scheduling problems, whose main elements can be recognized as the following:

Activities: $A = \{a_1, ..., a_n\}$ is the set containing the tasks. All the tasks have to be executed in order for the schedule to be completed. Every activity is characterized by a processing time p_i.

Resources: $R = \{r_1, ..., r_m\}$ is the set containing the resources required to execute the activities. Execution of each activity a_i can require an amount req_{ik} of resource r_k (k = 1,...,m) during its processing. There are different kinds of resources: disjunctive or cumulative, renewable or consumable, among others.

Constraints: The constraints are rules or restrictions that limit the possible allocations of the activities. They can be divided in two types: (1) the *resource constraints* limit the maximum capacity of each resource. For example, there may only be a certain number of machines or people available to work on some activities at any given time. (2) the *temporal constraints* impose limitations on the times at which activities can be scheduled. A unary constraint restricts a single activity, usually with a release time or a deadline. A binary constraint is imposed between to activities, for instance in order to mutually bind the instant of occurrence of their start times.

Because it has so many real-world applications, the scheduling problem has been widely studied by many scientific communities, such as the Artificial Intelligence (AI), Management Science (MS), and Operations Research (OR) community.

Nevertheless, these different approaches share a common drawback: they tend to neglect the fundamental aspect represented by the need to execute the found solutions in real working environments, where a variety of possible events may invalidate the current schedules making some proper and quick adjustments necessary. All this considered, we feel the need to introduce a broader definition of scheduling problem, consisting in the following two components:

- 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. Obviously, this sub-problem represents the commonly known scheduling problem;

- the **dynamic sub-problem**: it consists in monitoring the actual execution of the schedule and repairing the current solution (or producing brand new solutions), every time it is deemed necessary. The need to revise the schedule arises as a consequence of *exogenous event* occurrences which may eventually disturb the execution.

In this paper we provide an analysis of the dynamic sub-problem, we identify a number of particularly meaningful events which are likely to occur during schedule execution, and, finally, we show how these events may represent the building blocks for the construction of a benchmark generator.

3 Schedule execution and exogenous events

One of the fundamental aspects of any experimental analysis resides in the utilization of benchmarks which are independent from the particular strategy used to solve the core scheduling problem. The production of a schedule should be always performed in the light of subsequently putting it under execution in a real working environment. In a static world, synthesizing a conflict-free sequence of actions which satisfy a set of optimization criteria would be just enough, as in no execution the consistency of the found solution could ever be spoiled. In a non-static environment though, the perspective changes in that the validity retained by each schedule is inevitably destined to be only temporary.

Depending on the techniques used to solve the core scheduling problem, we can obtain solutions which exhibit different capabilities to absorb the effects of unexpected changes in the execution environment (different degree of schedule robustness) [7, 3, 8], though in general, the ability to produce robust solutions can not fully eliminate the necessity of rescheduling actions.

The synthesis of a benchmark requires a careful analysis of the most significant characteristics of the problem of interest. In the case of the scheduling execution problem, these characteristics are represented by the type, quality and number of unexpected events which can spoil the execution of the solution. In other words, the aspects of uncertainty which normally permeate the physical environments will have to be properly modeled through the characterization of a set of exogenous events of particular significance; the benchmarks we are pursuing to develop will be based on the production of sequences of elements taken from this set. Such uncertainty can be singled out in the following bullets:

activity delay, e.g., a surgery operation must be delayed until the doctors arrive;

growth of activity processing time, e.g., getting a flat tyre inevitably extends the duration of the journey;

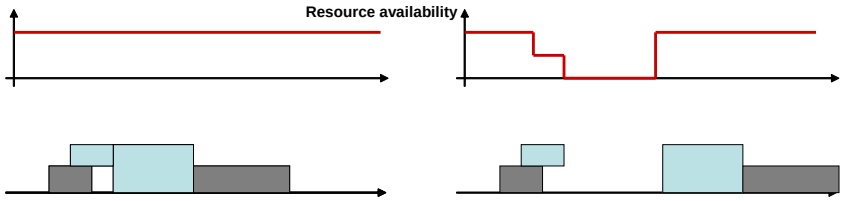
lowering of resource availability, e.g., unexpected loss of a piece of machinery in an assembly line;

variations in the number of activities, e.g., adding a visit to the mother in law in the daily schedule;

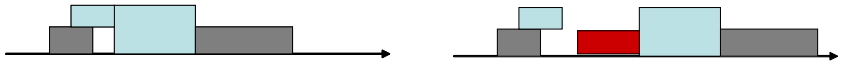
change in the mutual ordering of the activities, e.g., an activity in a production chain may suddenly become more urgent than another.

Therefore, the elements of uncertainty which may normally affect the consistency of a schedule basically belong to one of the following types: (1) temporal changes, which involve the various 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

(a) Temporal changes: activities can last more than expected or they can be postponed until necessary conditions are satisfied.



(b) Resource changes: the two red curves represent the nominal (left) and the actual (right) resource availability. The reduction of resource availability blocks the execution of the last two activities, which are delayed.



(c) Activity changes: the need of serving a new activity requires a reallocation of the current scheduled activities.

(d) Causal changes: a new precedence relation between a pair of activities requires a revision of previous choices.

Figure 1: On the same initial solution (left hand), different events may require interventions to re-establish the validity of the schedule

the activities. Moreover, during the execution of the schedule, the number of activities to be served may dynamically vary. Figure 1 shows how the events described above can affect the schedule during its execution.

3.1 The ingredients of the benchmark sets

In the last decades, several problem generators for project scheduling problems have been presented in scheduling literature [5, 6, 9]. Even though the presence of these generators represents an important boost for the development of new approaches to scheduling, their attention is limited on producing problem instances for the off-line step of the scheduling problem; in other words, they provide static instances for the sole aim of computing solutions that optimize one or more objective functions.

In order to define a benchmark set for the dynamic sub-problem, we refine here the event concept introduced above. For each exogenous events we provide in the following a detailed definition.

1. delay of an activity, e_{delay}:

$$e_{delay} = \langle a_i, \Delta_{st}, t_{aware} \rangle$$

besides the activity to be delayed a_i and the width of the shift, Δ_{st}, it is necessary to specify the instant where the specific event is supposed to happen, t_{aware} (this is a common element of all the defined events);

2. change of an activity duration, e_{dur};

$$e_{dur} = \langle a_i, \Delta_{dur}, t_{aware} \rangle$$

like the previous case it is necessary to specify three different parameters: the activity a_i, the change in duration Δ_{dur}, and t_{aware};

3. change of a resource availability, e_{res};

$$e_{res} = \langle r_j, \Delta_{cap}, st_{ev}, et_{ev}, t_{aware} \rangle$$

in this case, there are more parameters to specify: the resource involved r_j, the variation in resource availability Δ_{cap}, the time interval in which the change takes place $[st_{ev}, et_{ev}]$, and t_{aware}. We note that the time interval can be infinite, i.e. $et_{ev} \rightarrow \infty$;

4. change of the set of activities to be served, e_{act}

$$e_{act} = \langle \mu_a, a_k, \overline{req_k}, dur_k, est_k, let_k, t_{aware} \rangle$$

where the parameter $\mu_a \in \{add, remove\}$ is a flag that describes whether the activity a_k has to be added or removed; $\overline{req_k} = \{req_{k1}, \ldots req_{km}\}$

is a vector that define the resource requirement for each resource. Then the activity duration dur_k, the time interval in which this activity has to be served $[est_{a_k}, let_{a_k}]$ and t_{aware}. Of course we have that $let_k - est_k \geq dur_k$.

5. insertion or removal of a causal constraint between two activities, e_{constr}

$$e_{constr} = \langle \mu_c, a_{prec}, a_{succ}, d_{min}, d_{max}, t_{aware} \rangle$$

where the flag $\mu_c \in \{add, remove\}$ describes if the constraint between a_{prec} and a_{succ} has to be posted or removed. We need also to specify the minimum and maximum distance d_{min}, d_{max} imposed by the constraint and t_{aware}.

In case the Δ_{st} parameter of the e_{delay} event should be negative, this is reflected in starting the activity earlier than expected. Similarly, a negative value for Δ_{dur} in the e_{dur} event determines an early stop of the activity, while a negative value of Δ_{cap} determines an increase of the resource availability during the interval $[st_{ev}, et_{ev}]$.

Regarding the last two events, we have to say that in case of activity and/or constraint removal (e_{act} and e_{constr}) it is necessary only to specify the parameters related to the involved activities, that is a_k and the pair (a_{prec}, a_{succ}), respectively.

t_{aware}. In this section we have highlighted that there is an element which is present in every parameter list of all the different events: t_{aware}. This element specifies the instant where the specific event is supposed to happen. By using the t_{aware} parameter it is possible to temporally sort all the generated events and to tackle them in order of occurrence.

For instance, let's suppose that the two following events are generated: $e_{delay} = \langle a_1, 5, 13 \rangle$ and $e_{dur} = \langle a_2, 6, 7 \rangle$, describing respectively a 5 time units delay on the start time of activity a_1 to occur at $t_E = 13$ and a 6 time units increase on the duration of activity a_2 to occur at $t_E = 7$. Clearly, as the simulated execution starts, the two events will be ordered according to their occurrence time, that is:

1. $t_E = 7 \rightarrow e_{dur}$
2. $t_E = 13 \rightarrow e_{delay}$

In this example, at $t_E = 7$ the duration of activity a_2 will be increased, and, after counteracting the possible inconsistencies introduced by the event, the schedule will undergo the second occurrence at $t_E = 13$.

As will be explained in the next section, the synthesis of proper values for the parameter t_{aware} guarantees that all the events produced by the benchmark generator will be applicable to the schedule representation at all times.

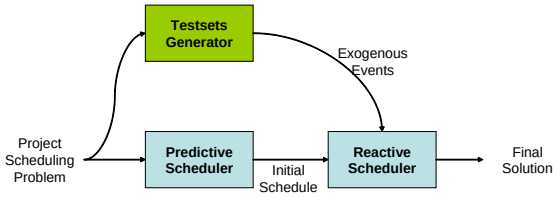


Figure 2: Reactive Scheduling Framework

4 A first benchmark generator

In this section we describe a first framework to generate benchmark data sets. As described above, we want to take into account the scheduling problem considering both the production of an initial solution (the static sub-problem) and the management of the actual execution of the solution (the dynamic sub-problem). Although, the general scheduling problem requires to create benchmarks for both sub-problems, at this first stage we focus our attention exclusively on the production of benchmarks for the dynamic scheduling sub-problem. In fact, as mentioned above, different benchmark generators for the commonly known scheduling problem have been deeply studied in literature [5, 6, 9]; therefore the benchmark generator will be based on scheduling problem instances already defined with these approaches. Figure 2 depicts the elements of an empirical framework for schedule execution, showing how the benchmark generator for the reactive scheduling problem is strictly decoupled from the scheduling problem solutions.

In the remainder of this section we first describe the scheduling problem we refer to, RCPSP/max[1], then we introduce the related benchmark generator.

4.1 RCPSP/max

The Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max [1], is here adopted as a reference problem. The basic elements of this problem are a set of *activities* denoted by $A = \{a_1, a_2, \ldots a_n\}$. Each activity has a fixed *processing time*, or *duration*, p_i and must be scheduled without preemption.

A *schedule* is an assignment of start times to activities $a_1, a_2, \ldots a_n$, i.e. a vector $S = (st_1, st_2, \ldots, st_n)$ where st_i denotes the start time of activity a_i. The time at which activity a_i has been completely processed is called *completion time* and is denoted by et_i. Since we assume that processing times are deterministic and preemption is not permitted, completion times are determined by $et_i = st_i + p_i$. Schedules are subject to both *temporal* and *resource constraints*. In their most general form temporal constraints designate arbitrary minimum and maximum time lags between the start times of any two activi-

[1] A generator for this problem is described in [9].

ties, $l_{ij}^{min} \leq st_j - st_i \leq l_{ij}^{max}$ where l_{ij}^{min} and l_{ij}^{max} are the minimum and maximum time lag of activity a_j relative to a_i. A schedule $S = (st_1, st_2, \ldots, st_n)$ is *time feasible*, if all inequalities given by the activity precedences/time lags and durations hold for each start time s_i. During their processing, activities require specific resource units from a set $R = \{r_1 \ldots r_m\}$ of resources. Resources are *reusable*, i.e. when they are released if no longer required by an activity, they are immediately available for use by another activity. Each activity a_i requires the presence of req_{ik} units of the resource r_k during its processing time p_i. Each resource r_k has a limited capacity of c_k units. A schedule is *resource feasible* if for each instant t the demand for each resource $r_k \in R$ does not exceed its capacity c_k, i.e. $\sum_{st_i \leq t < e_i} req_{ik} \leq c_k$. A schedule S is called *feasible* if it is both time and resource feasible.

4.2 The production of exogenous events for RCPSP/max

Section 3.1 has introduced different types of events which represent the ingredients of a benchmark data set. In this section we describe the input data that have to be provided to the benchmark generator. To this aim, it is possible to single out the following items:

- the scheduling problem P. This is necessary to have a knowledge of all the components: activities, resources, constraints and how they are related among each other.
- Number of events to generate.
- Probability of occurrence for each single type of event e:
  $$0 \leq P_e \leq 1$$
  such that $\sum P_e = 1$.
- The minimum and maximum magnitude of each type of event.

As mentioned above, a key point in a benchmark instance for the dynamic sub-problem is the fact that the different events have to be properly spaced in time. This aspect has been taken into consideration with the definition of the parameter t_{aware}, whose different values determine the instants where each specific event is supposed to happen. Finding a value for the t_{aware} parameter which is consistent for all possible executions is trivial only in the case of the event e_{res}: in fact, in this case, the condition $t_{aware} \leq st_{ev}$ can always be verified. In the other cases, as more activities are involved, it is in general not possible to define a consistent value of t_{aware} unless a deep analysis of the scheduling problem is done. This is due to mainly two reasons: (1) the solution of a scheduling problem is in general not unique, and (2) the start times of the schedule activities may decrease during execution because of task anticipations. For instance, if we need to delay an activity, obviously the value of t_{aware} should not to be greater than the start time of the activity; but since the start time of each activity is in general different according to different schedules there is no way to compute a set of t_{aware} values which are guaranteed to be valid for every possible execution unless some new hypothesis are introduced.

To compute consistent t_{aware} values and, then, overcome the difficulty pointed out above, we used a relaxed version of the scheduling problem in which resource constraints are not taken into account. This relaxed problem consists in a Simple Temporal Problem, or STP. An STP can be represented by CSP in which every constraint is binary (involves at most two variables) and a consistent solution is obtained, after a complete propagation, picking the lower admissible value for each activity – earliest start time solution. For a thorough discussion of STP the reader can refer to [4].

Therefore, using the relaxed version of the scheduling problem it is possible to compute the lower and the upper bound for the start and the end time of each activity, values that can be used to define the parameter t_{aware} for the events, at least for the initial solution. In fact, the occurrence of the produced events can make these bounds no longer valid (for instance if an activity duration is reduced and/or an activity is anticipated). For this reason we need to add a set of simplifying assumptions on the events that have to be generated:

- activities cannot be anticipated, that is, for each e_{delay} is $\Delta_{act} > 0$;
- activity durations can only increase, that is, for each e_{dur} is $\Delta_{dur} > 0$;
- there are only reduction of resource availability, that is, for each e_{res} is $\Delta_{cap} > 0$.

These assumptions allow to rely on the lower bounds of the start times of each activity, $lb(st_i)$, to define "safe" values for the t_{aware} parameter related to the different events. More precisely, we have that:

- in the case of a delay on activity a_i (e_{delay}), we assume that $t_{aware} \leq lb(st_i)$,
- in the case of change of duration of the activity a_i (e_{dur}), we assume that $t_{aware} \leq ub(et_i)$,
- in the case of adding/removing a new activity a_k (e_{act}), we assume that $t_{aware} \leq lb(st_k)$ if the activity a_k is removed, and $t_{aware} \leq est_k$ otherwise;
- in the case of adding/removing a new constraint between the two activity a_{prec} and a_{succ}) (e_{constr}), we assume that $t_{aware} \leq lb(st_{prec})$ if the constraint is removed, and $t_{aware} \leq min(lb(st_{prec}), lb(st_{succ}))$ otherwise.

Furthermore, these bounds can be also used to set other parameters of the different events:

- for the width of the delay on activity a_i (e_{delay}), we have that $\Delta_{st} \leq ub(st_i) - lb(st_i)$,
- for the change of activity duration (e_{dur}), we have that $\Delta_{dur} \leq ub(et_i) - lb(st_i) - p_i$, and
- for the change of resource availability (e_{res}), we have that $0 \leq \Delta_{cap} \leq cap_j$.

Example. To make the approach clear, we present an example of benchmark instance for a scheduling problem of eight activities (see Fig. 3), representing a multiple capacitated job shop scheduling problem instance. Each activity is characterized by a certain duration and requires one instance of one of the two available resources r_1 and r_2. Each resource has a maximum capacity $c_i = 2$; more precisely, oddly numbered activities require one instance of r_1 while evenly numbered activities require one instance of r_2. All the oddly numbered activities have a star-time of at least 3 (in Fig. 3 this fact is represented by the constraints, labeled with a value 3, between the source and the activities). The activity processing time vector is equals to $D = \{d_1, \dots d_n\} = \{4, 7, 4, 7, 3, 5, 3, 5\}$. Finally, the problem is also defined by the following precedence constraints $a_1 \prec a_2$, $a_3 \prec a_4$, $a_5 \prec a_6$, and $a_7 \prec a_8$ (i.e., activity a_6 can start only after a_5 ends).

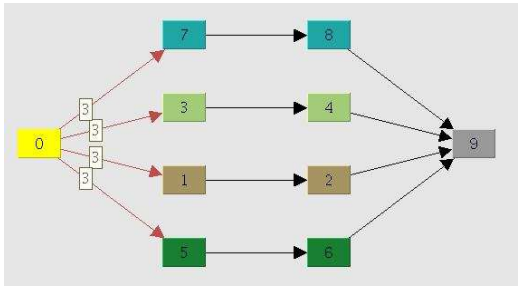
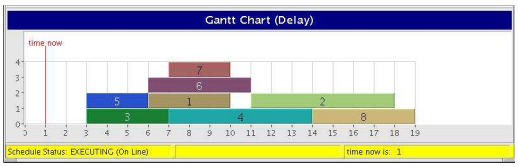


Figure 3: The schedule obtained as a consequence of the first event.

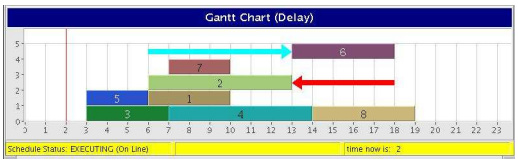
```
--------------------------------
 2
 {eventDelay  a6  7  2}
 {eventDuration  a2  5  4}
--------------------------------
```

Figure 4: An exogenous events instance.

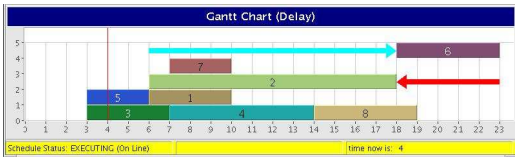
Figure 4 shows a possible instance produced by our benchmark generator, describing two exogenous events. The first event represents a delay related to the activity a_6: we have that $lb(st_6) = 6$ and therefore, a consistent value for t_{aware} is generated ($t_{aware} = 2$). The second event represents a change in the duration of a_2: the event will be acknowledged at $t_{aware} = 4$ and the activity duration will be augmented by 5 time units.



(a) The initial schedule.



(b) The schedule obtained as a consequence of the first event.



(c) The schedule after the second event.

Figure 5: Graphical visualization of the execution of a schedule for the problem in Fig. 3.

Figure 5 presents a graphical visualization of a possible execution of a schedule for the problem in Fig. 3. We extract these figures from the scheduler framework OOSCAR [2]. We note that the vertical red line in each figure represents the current execution time t_E.

Figure 5(a) shows a solution of the scheduling problem while Fig. 5(b) shows the solution found by the rescheduler after the first event ($t = 2$): the blue arrow represents the delay which affected a_6, while the red arrow represents the best action found by the solver (i.e. anticipating a_2) with the double aim of avoiding any conflict on resource usages and keeping the schedule makespan to a minimum.

To complete the example, Fig. 5(c) shows the effects of the second produced event ($t = 4$): the duration of activity a_2 is augmented and the solver therefore opts to further delay activity a_6 because activities a_2, a_6, and a_8, using the same resource, cannot simultaneously overlap.

5 Final remarks and ongoing work

The benchmark generator we are presenting here represents only an initial step: in order to design a complete experimental framework, it is essential to introduce also a set of metrics with the twofold aim of (a) evaluating the validity of the different rescheduling techniques by producing an assessment of the quality of the solution according to various criteria and (b) having a measure to assess the difficulty of the sets of generated events: the metrics

belonging to this set will be mainly used to bias the generation of the benchmark sets depending on their inherent difficulty.

Like for static scheduling, reactive scheduling is carried on with the goal of optimizing determined objective functions and/or satisfying a number of preferences. The "dynamic" optimization criteria are in general different then those related to the static case, as schedule execution imposes the presence of different requirements. Moreover, given the generally strict time availability over which the schedule revision procedure is called to react, sometimes solution quality must come as a secondary priority as the execution of the schedule does not allow for time-intensive computations.

Depending on the specific properties which have to be maintained, a number of metrics can be introduced in order to guide the re-scheduling procedure toward the most desirable solution. For instance, if the minimization of the schedule makespan (i.e., the total completion time) is the key aspect, the optimization function is trivially represented by the makespan itself. The schedule revision procedure will then be focused on keeping such measure at a minimum after each occurrence of disturbing events, thus maximizing the degree of optimality of the perturbed schedule at all times. More precisely, given t_E equal to the actual execution time and $T = \{a_i \mid st_i > t_E\}$, each re-scheduling action will involve each activity $a_i \in T$ and will be performed so as to minimize the schedule total completion time.

Another important measure is represented by the schedule *continuity* (or stability), which may informally be described as the closeness of the perturbed schedule to the schedule before the occurrence of the disturb. Solution continuity can be a very important quality measure of the schedule: in many cases it is in fact essential that any revised solution be as close as possible to the previous consistent solution found by the scheduler; the closer any two solutions are to each other, the higher their level of continuity.

As mentioned above, in order to properly bias the benchmark generator during the the synthesis of the appropriate sets of events, it is also essential to define a means of assessing the "gravity" of each event. These metrics have to be strictly related to the structure of each initial problem, as obviously, the same event may have enormous consequences on one specific schedule and little or no consequence at all on another. It is not the aim of this paper to present the details of such metrics; the formal synthesis of all the necessary definitions is the object of currently ongoing work and will be presented in future publications.

6 Conclusions

Our research work aims at analyzing and producing benchmark data sets for the scheduling execution problem. This effort is justified by the absence of such benchmarks and by our conviction that an experimental analysis of the execution of schedules is invaluable to assess the effectiveness of different approaches to scheduling problems.

To this aim we propose a benchmark based on the production and firing of a variable number of events chosen from a predetermined set, aimed at testing the effectiveness of rescheduling algorithm as well as the robustness of the initial schedule.

Future work will necessarily be dedicated to the introduction of a number of suitable metrics to realize a general empirical framework for the reactive scheduling problem.

REFERENCES

[1] 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.

[2] A. Cesta, G. Cortellessa, A. Oddi, N. Policella, and A. Susi. A Constraint-Based Architecture for Flexible Support to Activity Scheduling. In *In Proceedings AI*IA 01, LNAI N. 2175*, 2001.

[3] A. J. Davenport, C. Gefflot, and J. C. Beck. Slack-based Techniques for Robust Schedules. In *Proceedings of the 6^{th} European Conference on Planning, ECP'01*, 2001.

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

[5] E. L. Demeulemeester, B. Dobin, and W. Herroelen. A random activity network generator. *Operations Research*, 41:972–980, 1993.

[6] R. Kolish, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problem. *Management Science*, 41:1693–1703, 1995.

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

[8] N. Policella, S. F. Smith, A. Cesta, and A. Oddi. Generating Robust Schedules through Temporal Flexibility. In *Proceedings of the 14^{th} International Conference on Automated Planning & Scheduling, ICAPS'04*, pages 209–218. AAAI, 2004.

[9] C. Schwindt. Generation of Resource-Constrained Project Scheduling Problems Subject to Temporal Constraints. Technical report, WIOR-543, Universitat Karlsruhe, Germany, November 1998.