Designing a Testset Generator for Reactive Scheduling

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

Abstract. In this work we design 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 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 fact that the resolution of a scheduling problem consists both in the synthesis of an initial solution ("static" or "predictive" scheduling) and in the utilization of a number of methodologies dedicated to the continuous preservation of solution consistency (and quality). In fact, the occurrence of exogenous events during the execution phase in real working environments, often compromises the schedule's original characteristics.

Keywords: Scheduling, Benchmark Generation.

1 Introduction

The validity of a schedule is often very short. Scheduling is defined in theory as the problem of assigning a set of activities (or tasks) subject to a number of constraints, but the synthesis of initially feasible schedules is hardly ever sufficient; in real-world working environments, unforeseen events tend to quickly invalidate the schedule predictive assumptions and bring into question the continuing consistency of the schedule's prescribed actions. Therefore, the definition of scheduling problem has to be broadened so as to take into account both a "predictive" scheduling phase, whose aim is to propose a possible solution, and a "reactive" scheduling phase, whose objective is to maintain the quality of the current solution at execution time.

While the predictive scheduling aspects have been thoroughly evaluated through the production of several benchmarks and metrics, the aspect related to reactive scheduling has not yet received the same level of attention. As a matter of fact, the predictive and the reactive aspects of the scheduling problem are inherently correlated and cannot be separated; in other words, it is not possible to measure the quality of a rescheduling action if no information is given about which characteristics of the initial solution should be maintained, as well as it is not possible to assess the validity of an initial solution unless it has undergone a number of proper execution-time tests.

According to this point of view, our research work aims at producing a general framework for scheduling problems. In the course of our work we focus on the production of a Reactive Scheduling Testset Generator, as we recognize this as being a necessary instrument to assess the validity of the various rescheduling methodologies. The presence of this benchmark should reveal crucial to boost research on reactive scheduling and therefore, on general scheduling.

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. In 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**. $A = \{a_1, \ldots, a_n\}$ is the set of activities or tasks. Every activity is characterized by a processing time p_i.

– **Resources**. $R = \{r_1, \ldots, r_m\}$ 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 during its processing. There are different kinds of resources: disjunctive or cumulative, renewable or consumable, among others.

– **Constraints**. The constraints are rules 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 in which activities can be scheduled. 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). Yet, 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 [1]. All this considered, it is fundamental to introduce a broader definition of scheduling problem, consisting in the following two components:

- the **static sub-problem**: given a set of *activities* 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, every time it is necessary. The need to revise the schedule arises as a consequence of *exogenous event* occurrences.

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 reactive scheduling benchmarks.

3 Schedule execution and exogenous events

The synthesis of a benchmark requires the identification of the most significant characteristics of the problem. In the case of the reactive scheduling problem, the uncertainty aspects which normally permeate the physical environments will have to be properly modeled through the characterization of a set of particularly significant exogenous events; the benchmarks we are pursuing to develop will be based on the production of sequences of elements taken from this set. Real world 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 a 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 an unscheduled visit to the mother in law in the daily plan; *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 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.

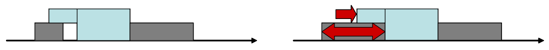
3.1 The ingredients of the benchmark sets

In the production of a benchmark set for our problem fundamental characteristics are represented by the type of unexpected events which can spoil the execution of the solution, their quality (or magnitude), and the way they are spaced in time. While the importance of the first two aspects is evident, the last point requires a further remark: to "simulate" a real executional behavior we need to know also when given events will happen: for instance let us suppose to have an activity scheduled at $t = 21$ and, suddenly, we know at $t = 13$ that this activity has to be delayed of 3 time-units. In this case we will have $21 - 13 = 8$ time units to compute a new solution for our current problem.

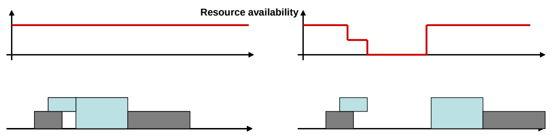
In this section we discuss how these aspects have been faced during the production of our testset generator.

The need of spacing the events. We have highlighted above the need of spacing the different exogenous events of the benchmark. This will be done introducing in the definition of the single type of the parameter t_{aware}. This element specifies the "absolute" 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 "fire" them in order of occurrence. For instance, let's suppose that the two following events are generated: 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. 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 event occurrence at $t_E = 13$.

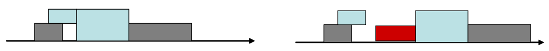
Why choosing absolute values for t_{aware}? An alternative could be to define it as a variable whose values are event-bound, for instance related to the start times of the activities. However, in this way the single instance will yield different executional behaviors w.r.t. the considered schedule (a problem can admit several solutions). A further reason is due to feasibility issues. In fact binding t_{aware} values to the start time of one activity could in fact produce a situation where an event, supposed to be fired on one activity at t_{aware}, may not be introduced because its value falls into the past w.r.t. the current execution time (see section 4.4 for an example). As will be considered in the next section, the fact that the t_{aware} parameter takes temporally "absolute" values, poses the problem of synthesizing t_{aware} values that are guaranteed to be valid for any possible schedule execution. In other words, it is necessary to make sure that all the events produced by the benchmark generator will be applicable to the schedule representation



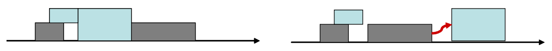
(a) Temporal changes: activities can last more than expected or they can be postponed until necessary conditions are satisfied.



(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, different events may require interventions to re-establish the validity of the schedule

at all times.

Definition of the different exogenous events. 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.

– *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 be detected, t_{aware} (this is a common element of all the defined events);

– *change of an activity processing time*, e_p;

$$e_p = \langle a_i, \Delta_p, t_{aware} \rangle$$

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

– *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. $et_{ev} \rightarrow \infty$;

– *change of the set of activities to be served*, e_{act}

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

where the parameter $f_a \in \{add, remove\}$ 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 $let_k - est_k \geq dur_k$.

– *insertion or removal of a causal constraint between two activities*, e_{constr}

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

where the flag $f_c \in \{add, remove\}$ 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_p in the e_p 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.

4 The Testset Generator

In this section we describe a 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

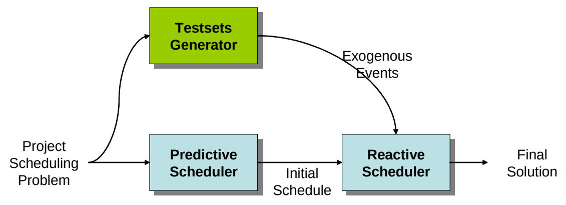


Figure 2: Reactive Scheduling Framework

dynamic scheduling sub-problem. In fact, as mentioned above, different benchmark generators for the commonly known scheduling problem have been deeply studied in literature [6, 7, 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, then we introduce the related benchmark generator.

4.1 The reference problem: RCPSP/max

The Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max [2], 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 interruption.

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 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 activities, $l_{ij}^{min} \leq st_j - st_i \leq l_{ij}^{max}$ 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 Modeling 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: (a) the scheduling problem $\mathcal{P}$, (b) the number of events to generate, (c) the probability of occurrence for each single type of event, and (d) 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 be detected. 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 proper 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 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 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 [5]).

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_p is $\Delta_p > 0$;
- only resource availability reductions are allowed, that is, for each e_{res} is $\Delta_{cap} > 0$.
- no causal constraint removals are allowed.

The previous assumptions guarantee that each event, regardless of its type and time of occurrence, will constrain the problem in a *monotonic* manner. In other words, given an initial problem $\mathcal{P}$ characterized by a constrainedness $\mathcal{C}$, and an event e_k, the problem $\mathcal{P}'$ obtained by adding e_k to $\mathcal{P}$, will be characterized by a constrainedness $\mathcal{C}' \geq \mathcal{C}$. This property of monotonicity, ruling out any chance of possible constraint relaxations, ensures that all the lower and upper bounds for the start and end times of each activity computed on the relaxed version of the scheduling problem, remain valid at all times. This allows to define "safe" values for the t_{aware} parameter related to the different events: (1) in the case of a delay of activity a_i (e_{delay}), we assume that $t_{aware} \leq lb(st_i)$, (2) in the case of change of duration of the activity a_i (e_p), we assume that $t_{aware} \leq lb(et_i)$, **(3)** 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; (4) in the case of adding/removing a new constraint between the two activities a_{prec} and a_{succ}) (e_{constr}), we assume that $t_{aware} \leq lb(et_{prec})$ if the constraint is removed, and $t_{aware} \leq min(lb(et_{prec}), lb(st_{succ}))$ otherwise (a causal link between two activities a_i and a_j is intended as a temporal constraint between et_{a_i} and st_{a_j}). Furthermore, these bounds can be also used to set other parameters of the different events: (5) for the width of the delay on activity a_i (e_{delay}), we have that $\Delta_{st} \leq ub(st_i) - lb(st_i)$, (6) for the change of activity duration (e_p), we have that $\Delta_p \leq ub(et_i) - lb(st_i) - p_i$, and (7) for the change of resource availability (e_{res}), we have that $0 \leq \Delta_{cap} \leq cap_j$.

4.3 Evaluating benchmark instances

In order to design a complete testset generator, it is essential to introduce a set of metrics with the aim of having measures to assess the difficulty of the sets of the generated events. In general, the same event may have enormous consequences on one specific schedule and little or no consequence at all on another solution. To overcome this drawback we consider in the evaluation of the single instance both the set of events and the problem on which this events will be applied. The idea is to use proper metrics (see below) to evaluate the structure of a scheduling problem as a set of unexpected events $\mathcal{E} = \{e_1, \ldots e_n\}$ is introduced. For instance, let us consider a scheduling problem $\mathcal{P}'$ obtained by adding to the original problem $\mathcal{P}$ an event e_k; given a metric $\mu()$, it is then possible to compare the structures of the problems $\mathcal{P}'$ and $\mathcal{P}$ by considering:

- the variation μ value: $\Delta_\mu = |\mu(\mathcal{P}) - \mu(\mathcal{P}')|$
- the speed of this variation, for instance: Δ_μ/Δ_t, with Δ_t equal to the distance between e_k and e_{k-1}

It is worth remarking that for the second aspect it is fundamental how the events are spaced over the horizon: given two events, the closer they are, the more critical the situation will be (note that the temporal distance between the events does not influence the overall variation). This corroborates the necessity to define t_{aware} values which are solution independent.

In the following paragraphs we describe scheduling metrics currently used to evaluate the benchmark instances. In particular these metrics face with the two main aspects of a scheduling problem: time and resource constraints.

Temporal metrics. The first metric we focus on quantifies the effects of the precedence constraints added in the plan of the tasks. To do this we start considering the notion of order strength described in [8]:

$$OS_{\mathcal{P}} = \frac{|\overline{\mathcal{P}}|}{n(n-1)/2} \qquad (1)$$

where $\overline{\mathcal{P}}$ denotes the set of precedence relations in the transitive closure of the precedence graph associated to $\mathcal{P}$, in other terms, $|\overline{\mathcal{P}}|$ denotes the number of activity pairs that are related. Thus, the lower the value of $|\overline{\mathcal{P}}|$, the more flexible the problem.

It is worth noting that the $OS_{\mathcal{P}}$ metric only gives a qualitative evaluation of the solution. In a problem like the RCPSP/max it is necessary to integrate this measure with another metric able to assess also the quantitative aspects of the problem (or solution). A possible metric that satisfies this requirement is defined in [4]. It requires the presence of a fixed-time horizon for the termination of all the activities. In order to compare two or more solutions we bound a single partial order schedule to have a finite number of solutions; then the metric is defined as the average width, relative to a given temporal horizon H, of the temporal slack associated with each pair of activities (a_i, a_j):

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

where $slack(a_i, a_j)$ is the width of the allowed distance interval between the end time of activity a_i and the start time of activity a_j. This metric characterizes the *fluidity* of a solution, i.e., the ability to use flexibility to absorb temporal variation in the execution of activities. Furthermore it considers that a temporal variation concerning an activity is absorbed by the temporal flexibility of the solution instead of generating deleterious domino effect (the higher the value of $fldt_H$, the less the risk, i.e., the higher the probability of localized changes).

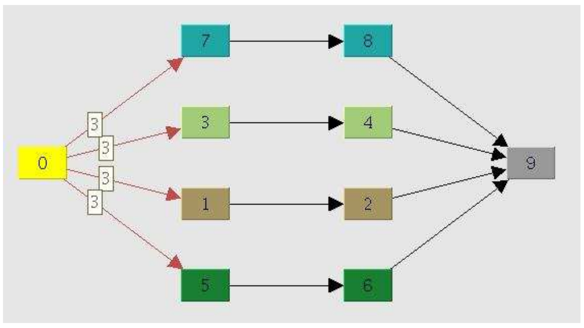


Figure 3: A graph representation of the problem.

Resource metrics. In order to understand the bias produced by a set of events on the resource-related characteristics of the scheduling problems, we employ another well-known measure, namely the resource strength. Given the resource r_k we denote with r^k_{min} the maximum usage of this resource by a single activity, $r^k_{min} = \max_{i=1..n} req_{ik}$. Also let r^k_{max} denote the peak demand of resource r_k in the earliest start schedule of the infinite capacity version of the problem. The resource strength of resource r_k is thus defined as:

$$RS_k = \frac{c_k - r^k_{min}}{r^k_{max} - r^k_{min}} \tag{3}$$

This metric takes into account the resource availability level with respect to the task requirements. Our analysis on the average RS over the resource which are employed in the scheduling problem $\mathcal{P}$, $RS_\mathcal{P}$. Like for $OS_\mathcal{P}$, the lower RS_k is, the less constrained the problem.

4.4 Example

To make the approach clear, we present an example of benchmark instances 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 start-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 $D = \{d_1, \ldots, d_n\} = \{4, 7, 4, 7, 3, 5, 3, 5\}$. 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., the evenly numbered activities can start only after the oddly numbered activities have terminated). The following list presents two possible exogenous events produced by our benchmark generator:

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

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.

t	0	2	4
$OS_\mathcal{P}$	0.14	0.14	0.14
$flex_{50}$	5.36	5.36	5.21
$RS_\mathcal{P}$	0.33	0.42	0.42

The table above shows the effects of the two events on the scheduling problem with respect the three metrics described in Sect. 4.3. We see how $flex_H$ and $RS_\mathcal{P}$ are able to catch the deteriorations produced by the two events ($OS_\mathcal{P}$ is not affected because the number of precedence constraints in the problem has not changed and no new activities have been added).

Figure 4 presents a graphical visualization of a possible execution of a schedule for the problem in Fig. 3 (we extracted the figures from the scheduling framework OOSCAR [3].). Note that the vertical red line in each figure represents the current execution time t_E. Figure 4(a) shows a solution of the scheduling problem while Fig. 4(b) shows the solution found by the re-scheduler 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. 4(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.

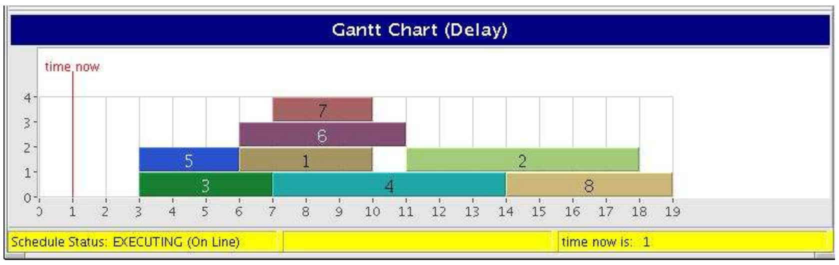
A slight modification to this example gives us the opportunity to illustrate the reason why choosing to produce absolute values for t_{aware} is the only viable option. Let us consider the following event sequence, where all t_{aware} values are expressed in relative terms:

```
{eventDelay  a6  7  st_6 - 4}
{eventDelay  a2  5  st_2 - 5}
```

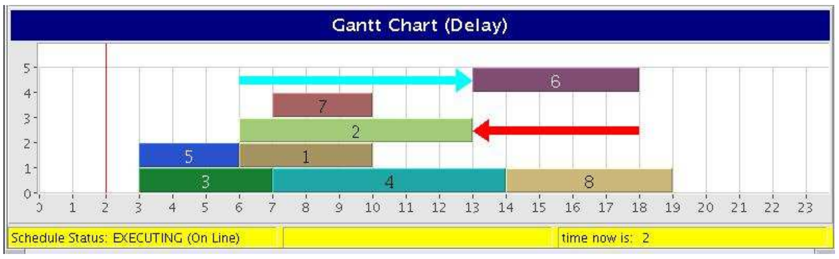
in this sequence, the first event will be fired at $t_{aware} = st_6 - 4 = 2$, producing the solution shown in Fig. 4(b); from here, we know that the second event should be fired at $t_{aware} = st_2 - 5 = 1$, but this is clearly impossible, as t_E is currently equal to 2 and t_{aware} can at no time be smaller than t_E.

5 Ongoing work

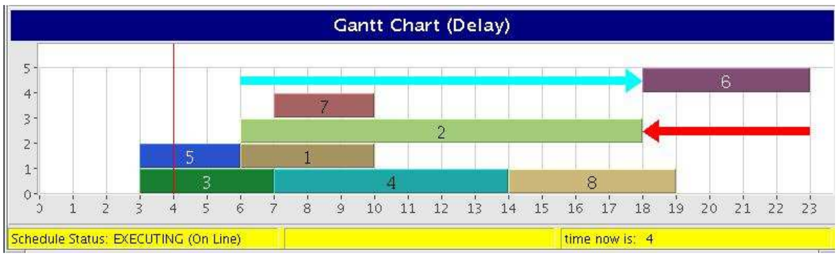
Our long-term plan concerns the realization of an experimental framework for scheduling problems (both static and dynamic). This is considered fundamental to evaluate the



(a) The initial schedule.



(b) The schedule after the first event.



(c) The schedule after the second event.

Figure 4: Graphical visualization of the execution of a schedule for the problem in Fig. 3.

integration of static and dynamic scheduling methodologies. At this stage we are working along two directions: the implementation of the benchmark generator and the analysis of a portfolio of reactive scheduling techniques. Regarding the benchmark generator we are currently in a tuning phase and we plan to make it available in a few months (see our web-site *http://pst.istc.cnr.it*). With regard to the scheduling techniques, we are evaluating the integration of predictive, flexible schedules together with localized repairing methods.

Furthermore, in order to design a complete experimental framework, it is essential to introduce also a set of metrics to evaluate the validity of the different rescheduling techniques by producing an assessment of the quality of the solution according to various criteria. The "dynamic" optimization criteria are in general different then those related to the static case, as schedule execution imposes the presence of different requirements. For instance, an 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. 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.

6 Conclusions

In this work we analyzed the main characteristics of a reactive scheduling problem with the aim of producing a benchmark generator. 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 proposed 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. In particular to guarantee the reproducibility of the experiment we have introduced the concept of t_{aware}, that is the instant in which a given event is detected. This allows to have instances that are independent from the initial solution of the scheduling problem.

7 Acknowledgments

This work was supported by the Italian Ministry of University and Scientific Research (MIUR) within the project “RoboCare: A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components”. The authors would like to thank Amedeo Cesta for his valuable help.

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, 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] 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.

[4] A. Cesta, A. Oddi, and S. F. Smith. Profile based Algorithms to Solve Multi Capacited Metric Scheduling Problems. In *Proceedings of AIPS-98*, pages 214–223, 1998.

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

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

[7] 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.

[8] A. A. Mastor. An Experimental and Comparative Evaluation of Production Line Balancing Techniques. *Management Science*, 16:728–746, 1970.

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

Nicola Policella is a Postdoctoral fellow at the ISTC-CNR. He received a Master degree and a PhD in Computer Science Engineering from the University of Rome “La Sapienza” in 2001 and 2005 respectively. In 2001, he was awarded the AI*IA prize for the best Master Thesis in AI. His current research interests include Scheduling with Uncertainty, Temporal and Resources Reasoning, and Constraint Programming.

Riccardo Rasconi is a PhD student in Information and Communication Technologies at the University of Genoa, a research fellow at the ISTC-CNR, and a AI*IA member. His research topics are focused on Reactive Scheduling, Temporal and Resource Reasoning, and Pervasive Computing.