Analysis of a Benchmark Generator for the Reactive Scheduling Problem

Amedeo Cesta
ISTC -CNR
Italian National Research Council
amedeo.cesta@istc.cnr.it

Nicola Policella
ESA/ESOC
European Space Agency
nicola.policella@esa.int

Riccardo Rasconi
ISTC -CNR
Italian National Research Council
riccardo.rasconi@istc.cnr.it

Abstract

This paper performs an analysis aiming at the production of a benchmark generator for Reactive Scheduling problem instances. The problem of monitoring the execution of a schedule and repairing it every time it is deemed necessary because of the action of unexpected exogenous events, is a rather hot topic in the research community; even more so, since the industry is increasingly becoming aware of the advantages that might be obtained through the use of dynamic schedule management systems, in terms of efficiency of the production lines and cost abatement.

Introduction

The validity of a schedule is often very short. Scheduling is defined in theory as the problem of assigning a start time to 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 consistency of the schedule's prescribed actions.

As a consequence, in order to be practically exploited in the real world, schedule management must be dealt with as a twofold process: the synthesis of an initial solution, which exhibits some desired characteristics ("static" or "predictive" scheduling), and the deployment of a number of methodologies dedicated to the continuous preservation of solution consistency and quality ("dynamic" or "reactive" scheduling). This second aspect of schedule management is necessary because of the inherent uncertainty that permeates the working environments where the produced schedules are normally expected to perform, uncertainty which eventually tends to spoil the schedule's original characteristics.

In order to fill the previous gap, a description of a benchmark generator instance is presented, along with a novel proposal suggesting how various metrics might be profitably employed to the twofold aim of either guiding the production of benchmark sets (basing on the difficulty assessment of the associated scheduling problem instances), and measuring the quality of the new solutions produced by each re-scheduling procedure, performed at schedule execution time.

The main objective of the research work we present aims at producing a *general* framework for scheduling problems: in the present paper 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 consequently, on general scheduling.

The Reactive Scheduling Problem: an Operational Interpretation

Because of its several 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 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 (Mc Kay, Safayeni, & Buzacott 1988; Aytug *et al.* 2005). All this considered, it is initially of fundamental importance to introduce a broader definition of scheduling problem which must comprise the following two aspects:

- the **static sub-problem**: given a set of *activities* (or tasks) and a set of *constraints*, it consists in computing a consistent assignment of start and end times for each activity. The solution of this problem is computed according to some optimization criteria, that depend on the quality measures of interest: a primary concern is often the total completion time (*makespan*), that should normally be kept as low as possible;
- 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[1].

Despite the limited level of attention received so far, in the very last years the reactive scheduling problem is undergoing an increasingly systematic study in the research communities: yet, in order to build a general experimental framework for reactive scheduling problem instances, the lack of

[1]Obviously, the static sub-problem aspect represents the commonly known Scheduling Problem, while the Reactive Scheduling Problem pertains to the dynamic sub-problem aspect of the extended scheduling problem definition.

a more detailed definition that takes into account both the qualitative and the quantitative aspects of the problem is evident.

The key idea is to define the reactive scheduling problem in terms of the disturbances injected to the base scheduling problem. Each disturbance will tend to modify the base problem's initial structure, within some extent: the difficulty of each reactive scheduling problem instance can therefore be assessed in terms of how much the base problem's initial characteristics have deviated from the initial values; the quality of the re-scheduling procedures can be assessed depending on how well they succeed in restoring/maintaining such desired characteristics in the solution as the outcome of the revision process. In this paper we therefore 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.

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

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

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

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

Very informally, a schedule under execution at time $t = t_E$ can be defined as a partition $\mathcal{V}(t) = \mathcal{V}_{\prec t_E} \cup \mathcal{V}_{\approx t_E} \cup \mathcal{V}_{\succ t_E}$ where $\mathcal{V}_{\prec t_E}$ is the set of the schedule activities that have already terminated, $\mathcal{V}_{\approx t_E}$ is the set of the activities are currently under execution, and $\mathcal{V}_{\succ t_E}$ is the set of the activities that have yet to begin, for $t_E = i, i \in \{0, 1, \ldots, \infty\}$.

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

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

The changes carried out by the modifiers to the interested Z set vary from the insertion and/or deletion of elements to/from Z, to simple alterations of the characteristics of those elements.

Based on the graph representation of a scheduling problem $P = G_P(V_P, E_P)$, we can exploit the notion of Instant Modifier to give an alternative definition of solution S:

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

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

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

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

One issue worth being remarked is that, as confirmed in Definition 3, both the scheduling problem instances P and their relative solutions $S \in \mathcal{S}(P)$ *share the same structure*. In fact, both can be formalized through a tuple $\langle \mathcal{V}, \mathcal{C}, \mathcal{R} \rangle$, the difference being in the number and/or allocation of the constraints $c_i \in \mathcal{C}$ that are necessary to eliminate the temporal and/or resource conflicts present in P.

Notice that a scheduling algorithm can be understood as the application of the modifier $mod_\star^C$. Notice also that $mod_\star^{C,k}$ extracts the k^{th} solution from P; in general in fact, a scheduling problem P admits many solutions.

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

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

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

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

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

The reader should notice that solving the RSP is different from solving an ordinary PSP, because of the presence of the time variable t. Being the RSP a dynamic problem, the time component plays an essential role in the solving process of the RSP instances. For the time being, it is important to highlight that, given a base scheduling problem P_{base} and three modification sets $\Delta_\mathcal{V}(t), \Delta_\mathcal{C}(t), \Delta_\mathcal{R}(t)$, the following tuples $\langle P_{base}, \Delta_\mathcal{V}(t), \Delta_\mathcal{C}(t), \Delta_\mathcal{R}(t), t_1 \rangle$ and

$\langle P_{base}, \Delta_{\mathcal{V}}(t), \Delta_{\mathcal{C}}(t), \Delta_{\mathcal{R}}(t), t_2 \rangle$ represent two different RSPs, as the mere passing of time inherently modifies the reallocation possibilities in case a re-scheduling is necessary. Intuitively, a re-scheduling action triggered by the occurrence of an exogenous event, may yield different results depending on the number of activities that have begun the execution or that have already terminated, and such number is obviously a function of time.

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

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

In the remainder of this work, the term "problem" will be interchangeably used in both the *scheduling* and *reactive scheduling* acceptation; though all efforts will be made to underscore the specific context we will be referring to, the reader is invited to pay particular attention to this important differentiation.

Schedule Execution and Exogenous Events

The previous analysis lays the foundations for the production of a usable benchmark for reactive scheduling problems, as it introduces a formalism that allows to approach schedule dynamic management from a quantitative standpoint. The uncertainty aspects which normally permeate the physical environments have in fact been taken into account through the concept of *instant modifier*, which can be considered as the general representation of an *executional event*. In what follows, we try to further specify such modifiers toward operational templates strictly related to the specific scheduling technology employed. The most relevant aspects of real world uncertainty will be therefore modeled through proper instantiations of such templates, and the benchmarks we are pursuing to develop will be based on the production of sequences of such instantiations.

In the realm of scheduling, real world uncertainty is often singled out in the following points: *activity delay*, e.g., waiting until all passengers summon might delay the beginning of a trip; *growth of activity processing time*, e.g., making many stops inevitably extends the duration of the journey; *lowering of resource availability*, e.g., breaking an engine part has repercussions on the planned line of action; *variations in the number of activities*, e.g., adding an unscheduled detour adds up a new task to be accomplished; *change in the mutual ordering of the activities*, changing the priority order of some activities might require to re-think the journey's plan.

The Components of the Benchmark Sets

In the production of a benchmark set for scheduling problems, a number of different issues must be properly analyzed. As we have said, one such issue is related to the identification of the type of unexpected events which can spoil

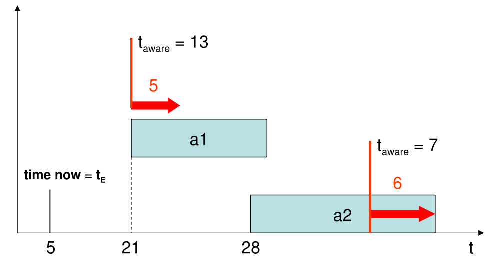


Figure 1: Sequential injection of two temporally spaced events.

the execution of the solution; another point is the introduction of methodologies to assess the difficulty of a benchmark instance. Yet, as the problem for which we are devising the benchmark is a dynamic problem, one more subtle point require attention: the temporal spacing among the events.

While the importance of the first two aspects is evident, the last point requires a further remark: as we have seen in Definition 2, in order to "simulate" a dynamic behavior we need to specify the exact moment in time when the given events will occur: the temporal variable t serves this purpose, and is of great importance to "control" the temporal aspects of the benchmark sets injection, as well as the urgency of the related reactive scheduling problem instance. In the next sections all the issues encountered in the design of our testset generator will be discussed in detail.

Temporal separation of the events Recalling Definition 2, we can see that the problem modifiers have been defined through the use of a temporal parameter to highlight the fact that such operators could be launched at any time during the execution of the schedule. In the operational use of the RSP benchmarks, a schedule is supposed to be executed and constantly repaired every time a new benchmark instance (event) is injected; hence, each event must be characterized with a parameter stating the instant in which the *execution simulator system* is supposed to acknowledge the event occurrence. This is done by introducing the parameter t_{aware} in the definition of each event, for any event type. It should be noted that the t_{aware} parameter specifies the "absolute" instant where the specific event is supposed to happen, and through its use, it is possible to temporally sort all the generated events and to "fire" them in order of occurrence. The reader should not mistaken the event occurrence time defined by the parameter t_{aware} with the temporal instant which involves the same event along the schedule timeline: the former defines "when" the event will take place; the latter defines "where" the event is localized. For instance, in Figure 1 the first event occurs at time $t_{aware} = 7$ but is localized at $t_{loc} = 28 + duration(a_2)$; the second event occurs at time $t_{aware} = 13$ but is localized at $t_{loc} = 21$; the temporal distance $t_{loc} - t_{aware}$ (computation time window) reveals a good evaluation of the urgency of an exogenous event.

As explained above, the t_{aware} parameter is essential to provide every exogenous event with a precise time of occurrence: but why choosing absolute values to define such parameter, as opposed to linking the t_{aware} variable to the affected activity's start or end time? There are mainly two reasons why a "relative" definition of t_{aware} cannot be used in practice: the first reason regards the necessity of generating reactive scheduling problem benchmarks whose difficulty is as much as possible independent from the particular re-scheduling technology employed; the second reason is related to feasibility issues: in fact, it is of primary importance that the dynamic model employed to represent the continuing schedule execution be reality-consistent at all times, which cannot be guaranteed if the t_{aware} values are problem-dependent.

Definition of the different exogenous events In order to define a benchmark set for the reactive scheduling problem, we refine here the concept of *Instant Modifier* ($mod^{Z}(t)$) previously introduced. Every modifier entails an alteration on the Z component of the base scheduling problem; depending on the particular nature of Z, each modification directly translates into a different type of exogenous event, of which we provide a detailed parametric definition. For all event types, the t_{aware} parameter is present with the usual meaning.

- the event e_{delay}, characterizing an activity delay, is defined as:

$$mod^{\mathcal{C}}(t) = \langle a_i, \Delta_{st}, t_{aware} \rangle \quad (1)$$

  where:
  - a_i is the activity affected by the delay;
  - Δ_{st} is the extent of the delay. Note that in general this quantity can be negative, in which case the activity is *anticipated*;

- the event e_{dur}, characterizing a change of an activity duration, is defined as:

$$mod^{\mathcal{C}}(t) = \langle a_i, \Delta_{dur}, t_{aware} \rangle \quad (2)$$

  where:
  - a_i is the activity whose duration is affected by the variation;
  - Δ_{dur} is the extent of the duration change (negative or positive);

- the event e_{res}, characterizing a change of a resource availability, is defined as:

$$mod^{\mathcal{R}}(t) = \langle r_j, \Delta_{cap}, st_{ev}, et_{ev}, t_{aware} \rangle \quad (3)$$

  where:
  - r_j is the resource involved in the event;
  - Δ_{cap} is the extent of the variation in resource availability (negative or positive);
  - $[st_{ev}, et_{ev}]$ represents the time interval spanned by the event. Note that in general this interval can have infinite width ($et_{ev} \rightarrow \infty$);

- the event e_{act}, characterizing a change of the set of the problem activities, is defined as:

$$mod^{\mathcal{V}}(t) = \langle f_a, a_k, \overline{req_k}, dur_k, est_k, let_k, t_{aware} \rangle \quad (4)$$

  where:
  - $f_a \in add, remove$ is a flag that describes whether the event is aimed at adding or removing the activity identified by a_k to/from the set $\mathcal{V}$;
  - $\overline{req_k} = \{req_{k1}, req_{k2}, \ldots, req_{km}\}$ represents the array that determines the requirements issued by a_k for all the m resources (required only if $f_a = add$);
  - dur_k represents the duration of a_k (required only if $f_a = add$);
  - $[est_k, let_k]$ represents the time interval in which a_k might be inserted (required only if $f_a = add$), where est_k and let_k are, respectively, the admissible *early start time* and the *latest end time* for a_k;

- the event e_{causal}, characterizing the insertion/removal of a causal constraint between two activities, is defined as:

$$mod^{\mathcal{C}}(t) = \langle f_c, a_{prev}, a_{succ}, d_{min}, d_{max}, t_{aware} \rangle \quad (5)$$

  where:
  - $f_c \in add, remove$ is a flag that describes whether the event is aimed at adding or removing the constraint existing between the activities identified by a_{prev} and a_{succ} to/from the set $\mathcal{C}$;
  - d_{min} and d_{max} represent, respectively, the minimum and maximum distance that the constraint must impose between a_{prev} and a_{succ} (required only if $f_c = add$).

The reader should notice that the e_{delay}, e_{dur} and e_{causal} events represent a specialization of the modifier $mod^{\mathcal{C}}(t)$; the e_{res} event is a specialization of the modifier $mod^{\mathcal{R}}(t)$, while the event e_{act} specializes the modifier $mod^{\mathcal{V}}(t)$.

The Generation of Exogenous Events

This section describes the general framework we devised for benchmark data sets generation. We have already mentioned that different benchmark generators for the static sub-problem have been deeply studied in literature (Demeulemeester, Dobin, & Herroelen 1993; Kolish, Sprecher, & Drexl 1995; Schwindt 1998); these efforts have resulted in the production of formally defined scheduling problem instances characterized by various levels of difficulty. The benchmark generator we propose will be based on the scheduling problem instances 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. The framework is composed of the following modules: (1) a Predictive Scheduler, which is in charge of solving the static sub-problem by synthesizing the baseline schedule; (2) the Reactive Scheduler, which receives in input the baseline schedule and solves the dynamic sub-problem by taking care of the solution maintenance during the execution; (3) the Testset Generator module, which produces the reactive scheduling benchmark sets.

Within the depicted framework, each benchmark set is treated as a second input to the Reactive Scheduler, which is called to acknowledge the events and revise the executing schedule according to the updated conditions. Note that, as Figure 2 shows, the PSP instances are a necessary input for Testset Generator, as the knowledge of the structure of each base scheduling problem instance P is necessary to synthesize exogenous events meaningfully related to P. Besides, this is in perfect accordance with Definition 2 of Instant

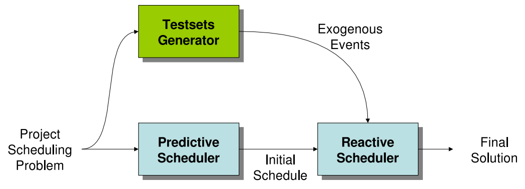


Figure 2: Reactive Scheduling Framework

Modifier, presented as an operator that applies to a problem instance P in order to alter some of its original properties.

In the next sections we focus on some important issues encountered during the process of tailoring the production of the benchmark sets to instances of our reference scheduling problem, the RCPSP/max (Bartusch, Mohring, & Radermacher 1988).

Timing the Exogenous Events consistently

A RSP benchmark generator is supposed to produce event instances whose characteristics (type, extent, etc.) are computed pseudo-randomly, to the aim of synthesizing benchmarks of a given difficulty. In the particular case of reactive scheduling, every benchmark instance I_{RS} must be generated relatively to one base scheduling problem instance I_S; the analysis of I_S is necessary to determine the bounds within which the randomization must operate in order to produce meaningful disturbances.

Given a baseline solution S_P of a scheduling problem P, the production of a set of disturbing events, each characterized by values of the t_{aware} parameter which are consistent for all possible executions of P's solutions, is not trivial for the following 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. Since it is not possible to know in advance all possible decisions taken by every rescheduler during the revision process, the new solution might in general present an allocation of the activities that does not allow a consistent introduction of the subsequent event present in the benchmark set.

In order to overcome this difficulty and therefore produce events characterized by t_{aware} values that are guaranteed to be valid for every possible execution, (a) we use a relaxed version of the scheduling problem in which we strictly focus on the temporal aspects of the problem (i.e. disregarding all the resource constraints), and (b) we introduce a number of conditions which guarantee that whatever the executional conditions and the time of occurrence, each produced event will exhibit a *positive computation time window*. This relaxed problem consists in a Simple Temporal Problem, or STP. An STP can be represented by simple temporal network (STN) and solved as a constraint satisfaction problem in which every constraint is binary and a consistent solution is obtained, after a complete propagation, picking the lower admissible value for each activity – Earliest Start Time solution (Dechter, Meiri, & Pearl 1991).

Figure 3 shows a scheduling problem composed of five activities and its associated simple temporal network. Depending on the network structure and on the values pertaining to the network constraints, every time point tp_i is associated to an interval $[lb_i, ub_i]$ of admissible values for tp_i. The width $ub_i - lb_i$ of each interval is inversely proportional to the “constrainedness” of the associated time point; in a network characterized by a temporal horizon H, every time point tp_i can be associated to an interval whose largest width is H ($ub_i = H$, $lb_i = 0$), corresponding to the lowest level of time point constrainedness. Obviously, the highest level of time point constrainedness is reached when $ub_i = lb_i$, in which case only one feasible value exists for tp_i.

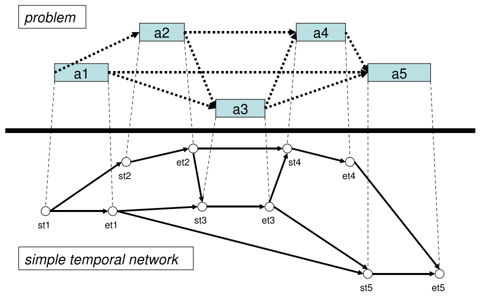


Figure 3: Simple Temporal Network underlying a Scheduling Problem

A determined level of constrainedness[2] $cstr$ can therefore be computed for any given STN, as follows:

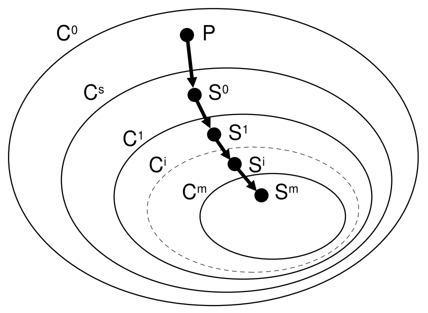
$$cstr = \frac{nH}{\sum_{i=1}^{n}(ub_i - lb_i)} \quad (6)$$

where n is the number of time points and H is the temporal horizon. Given a scheduling problem P and its related temporal network TN_P characterized by a constrainedness $cstr$, all the time points are therefore associated to a lowest and a highest possible value, respectively represented by the lower and the upper bound of their admissibility interval; this basically means that whatever the chosen time point $tp_i \in [lb_i, ub_i]$, all attempts to anticipate tp_i before lb_i or to delay it after u_i are destined to cause a temporal propagation failure, *unless some of the network constraints are retracted*.

If constraints can only be added and no problem constraint relaxations are ever allowed as new solutions are found during the execution, the constrainedness of the temporal network is bound to constantly increase (see Figure 4), and so are the lower bounds of all the admissibility intervals associated to every time point. At the activity symbolic level, this circumstance directly entails that an activity's start time can only be shifted rightwards with respect to the original position, therefore widening the computation time windows of all the exogenous events synthesized for that activity and keeping valid the t_{aware} values assigned to such events at all times.

Yet, certain types of exogenous events, like activity anticipations or processing time reductions, might still model situations that necessarily entail constraint retractions. At the present stage of our work, we therefore decided to work around a set of simplifying assumptions to be applied at benchmark generation time that, if verified, guarantee the

[2]Note that $cstr \in [1, \infty)$; if necessary, the reciprocal *looseness* ($lsns$) metric can be used, with $lsns = \frac{1}{cstr} \in [0, 1]$.



constrainedness: $C^0 \le C^s \le C^1 \le \ldots \le C^i \le \ldots \le C^m$

Figure 4: Constrainedness increasing with the occurrence of exogenous events

monotonic increase condition as shown in Figure 4, for the constrainedness of the temporal network associated to the scheduling problem instance P. Such assumptions do not represent "dogmatic" restrictions, they rather represent a necessary starting point in the analysis of reactive scheduling benchmarks. Current research is ongoing in order to provide for their possible relaxation; moreover, it should be noted that they are not too restrictive: the range of reactive scheduling problems that can be formulated, even in the presence of these assumptions, covers many practical and not trivial situations.

More specifically, we currently consider events that do not entail:

- activity anticipations:

$$\Delta_{st} > 0, \quad \forall e_{delay} = \langle a_i, \Delta_{st}, t_{aware} \rangle \tag{7}$$

- reductions of activity durations:

$$\Delta_{dur} > 0, \quad \forall e_{dur} = \langle a_i, \Delta_{dur}, t_{aware} \rangle \tag{8}$$

- activity removals:

$$f_a = add, \ \ \forall e_{act} = \langle f_a, a_k, \overline{req_k}, dur_k, est_k, let_k, t_{aware} \rangle \tag{9}$$

- causal constraint removals:

$$f_c = add, \ \ \forall e_{causal} = \langle f_c, a_{prev}, a_{succ}, d_{min}, d_{max}, t_{aware} \rangle \tag{10}$$

The previous assumptions facilitate the computation of precise temporal bounds, based on the temporal network underlying the original problem P. Under the *monotonic increase* condition, such bounds easily allow to define "safe" values for the t_{aware} parameter, each related to a different event type.

More precisely, let us suppose to have a scheduling problem P composed of a set of n activities $V = \{a_1, a_2, \ldots, a_n\}$ where every a_i is characterized by a start time $st_i \in [lb(st_i), ub(st_i)]$ and an end time $et_i \in [lb(et_i), ub(et_i)]$. For each event type, we can compute the following bounds:

- $[e_{delay}]$ In the case of delay of an activity a_i, we assume that:

$$t_{aware} \le lb(st_i) \tag{11}$$

- $[e_{dur}]$ In the case of change of a_i's processing time p_i, we assume that:

$$t_{aware} \le lb(et_i) \tag{12}$$

- $[e_{act}]$ In the case of insertion of an activity a_k, we assume that:

$$t_{aware} \le est_k \tag{13}$$

- $[e_{causal}]$ In the case of insertion of a new constraint[3] between the two activities a_{prev} and a_{succ}, we assume that:

$$t_{aware} \le min[lb(et_{prev}), lb(st_{succ})] \tag{14}$$

Quantifying the Benchmark Instances Consistently

The main reason that motivated the analysis of the events timings is *feasibility*; but consistent timing is just one aspect of event feasibility. The previous results, based on the temporal network underlying the scheduling problem, reveal extremely useful to the aim of establishing precise limits within which to perform randomization in the generation process of the reactive scheduling benchmark sets. Since the benchmark generation process is based on an aprioristic analysis of the initial temporal network, the knowledge of the temporal bounds that affect every time point is essential to produce events of meaningful size, i.e., events that do not uselessly overstress the network structural properties.

More precisely, the bounds that can be extracted from the scheduling problem are the following:

- $[e_{delay}]$ For the width of the delay of an activity a_i:

$$0 < \Delta_{st} \le ub(st_i) - lb(st_i) \tag{15}$$

- $[e_{dur}]$ For the change of a_i's processing time p_i:

$$0 < \Delta_{dur} \le ub(et_i) - lb(st_i) - p_i \tag{16}$$

- $[e_{res}]$ For the change of resource r_k's maximum capacity:

$$0 \le \Delta_{cap} \le C_k^{max} \tag{17}$$

The previous statements fix, for each event type of interest, the rigid bounds within which the size of the events must be randomly generated. For example, in the case of a delay of the activity a_i, the benchmark generator will produce an e_{delay} event whose Δ_{st} size takes a randomly generated value between 1 and $ub(st_i) - lb(st_i)$, as this quantity clearly represents the maximum allowed distance between st_i and et_i allowed by the temporal network.

Yet, choosing the event size between the previous bounds represents a necessary *but not sufficient* condition to guarantee event feasibility, for the following reason: given a scheduling problem P and a baseline solution S_P, they are generally characterized by two different temporal networks (TN_P and TN_S) both exhibiting different degrees of constrainedness ($cstr = \mathcal{C}^0$ and $cstr = \mathcal{C}^s$ respectively, with $\mathcal{C}^0 \le \mathcal{C}^s$); moreover, during the execution of the schedule, either the exogenous events and the consequent rescheduling processes tend to constantly increase the constrainedness of the current solution's temporal network according to the schema shown in Figure 4. Since the bounds

[3]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}.

defined by the equations 15, 16 and 17, are based on the analysis of P's network, which is characterized by the lowest $cstr = \mathcal{C}^0$ value, the events whose size is computed on the basis of such $cstr$ value cannot be guaranteed to model a consistent situation, as the network's constrainedness increases. In other words, the total dynamic acceptance of all the produced benchmark instances cannot be guaranteed: yet, a wise management of temporal constrainedness can be profitably used to control the benchmark difficulty, as shown in the next section.

Setting the Difficulty of the Benchmark Instances

In order to design a complete testset generator, it is essential to introduce a set of metrics to the aim of controlling the difficulty related to each generated event. The need to assess the impact factor of an event on the execution of a schedule raises from the fact that in general, the same event may have enormous consequences on one specific schedule and little or no consequence at all on another solution; for this reason, it is not possible to detach the reactive scheduling benchmark set from the scheduling problem instance it is intended to be applied to.

Depending on the particular executional aspect we are interested at, several metrics $\mu()$ can be used to synthesize events of determined size, that aim at interfering with that aspect by introducing specific changes of controllable gravity. Once these metrics are identified, they can therefore be used to bias the generation of the benchmark instances, in order to produce events characterized by different difficulty levels, with respect to the specific scheduling problem. The idea is to use the metric $\mu()$ to evaluate the structure of the scheduling problem as a set of unexpected events $\mathcal{E} = \{e_1, \dots, e_n\}$ are systematically introduced during its execution.

For instance, let us consider a scheduling problem P^k obtained by introducing the event e_k during the execution of the original problem P^0; apart from the particular aspect measured by the chosen metric $\mu()$, it is possible to compare the structural properties of the problems P^k and P^0, and therefore assess the event difficulty, in the following ways:

- by measuring the absolute variation with respect to the previous problem:
$$\Delta_\mu = |\mu(P^k) - \mu(P^0)| \tag{18}$$
- by measuring the speed of this variation:
$$\frac{\Delta_\mu}{\Delta_t} = \frac{|\mu(P^{k+1}) - \mu(P^k)|}{\Delta_t} \tag{19}$$
where Δ_t represents the temporal distance between e_{k+1} and e_k.

It is worth remarking that for the second aspect is fundamental how the events are spaced over the horizon: given two events, the closer they are, the more critical the situation will be. This corroborates the necessity to define t_{aware} values which are solution independent: In the following paragraphs we describe the metrics currently used to evaluate the benchmark instances. In particular, these metrics directly reflect the two main aspects that permeate the scheduling problem, namely, the aspects related to the temporal constraints and to the resource constraints.

Temporal metrics. Each of the temporal metrics we describe represents a particular interpretation, at the activity symbolic level, of the general metric previously defined as *constrainedness*. The different metrics will be used depending on the particular temporal aspect of interest.

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 (Mastor 1970):

$$OS_P = \frac{|\overline{P}|}{n(n-1)/2} \tag{20}$$

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

It is worth noting that the OS_P metric only gives a qualitative evaluation of the solution's constrainedness in terms of constraint structure. In a problem like the RCPSP/$\max$ it is often necessary to integrate this analysis with another metric, able to assess also the quantitative aspects of the current solution. A possible metric that satisfies this requirement is the fluidity $fldt$ (Cesta, Oddi, & Smith 1998). 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 bind 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):

$$fldt_H = \sum_{i=1}^{n} \sum_{j=1 \wedge j \neq i}^{n} \frac{slack(a_i, a_j)}{H \times n \times (n-1)} \times 100 \tag{21}$$

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, which can be interpreted as the potential to use the network's available *looseness* to absorb temporal variations during the execution of the activities.

Similarly, if the aspect we are interested at is solution *stability*, q suitable metric is the disruptibility $dsrp$ (Policella *et al.* 2004):

$$dsrp = \frac{1}{n} \sum_{i=1}^{n} \frac{slack_{a_i}}{num_{changes}(a_i, slack_{a_i})} \tag{22}$$

where $num_{changes}(a_i, slack_{a_i})$ computes the number of activities whose temporal position changes consequently to a delay of size $slack_{a_i}$ imposed on activity a_i.

The greater the difference Δ_μ caused by the insertion on the event, the greater the problem's loss of stability, the more difficult the problem faced by the re-scheduler to find an adjustment that minimizes such loss.

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* ((Schwindt

1998)), defined as follows:

$$RS_k = \frac{C_k^{max} - r_{min}^k}{r_{max}^k - r_{min}^k} \tag{23}$$

In which, given the resource r_k, we recognize the following elements:

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

For each resource r_k, this measure takes into account the resource availability level with respect to the task requirements. The metric $\mu()$ we are interested at, is based on the average value of RS_k over all the resources employed in the scheduling problem P:

$$RS_P = \frac{\sum_{k=1}^{m} RS_k}{m} \tag{24}$$

As opposed to the Order Strength OS_P, the higher RS_k is, the less constrained is the problem. Again, using the metric RS_P we can generate events that globally aim at reducing the "safety margin" between the resource maximum capacities and the average resource utilization, thus increasing the probability (if not directly causing the development) of contention peaks.

Conclusions

The present work aimed at analyzing the production of benchmark data sets for the scheduling execution problem. This effort is justified by the absence of such benchmarks in the scheduling literature, and by our conviction that they represent a necessary means to foster: (a) significant experimental analysis and (b) scheduling competitions, in the reactive scheduling problem area. The devised benchmarks basicly consist of a set of properly synthesized schedule modifications (exogenous events), aimed at simulating the environmental uncertainty; during the execution of the schedule, the events are iteratively introduced, in order to spoil some of the initial schedule's characteristics in a determined and measurable fashion.

To this aim, it is therefore essential to provide a precise definition of all the metrics of interest, in order to qualitatively and quantitatively assess the changes that a schedule may undergo during the execution in such unpredictable conditions (measuring the difficulty of each particular set of events). This analysis is performed on the basis of the initial scheduling problem structure, as the impact factor of a disturbance is a function of the particular schedule the same disturbance is applied to. The effectiveness of a rescheduling action will be obviously measured with respect to the gravity of the occurred modification.

In order to minimize the possibility of event rejection and guarantee an acceptable level of feasibility, the event's timing and size must be carefully determined during the benchmark production process, through a deep analysis of the scheduling problem. The assessment of consistent event timings and sizes is essential to avoid the production of useless benchmark instances, such as events that are related to activities that have already terminated, or events that stress the problem structure beyond its physical limits. Such analysis results in the introduction of a set of preliminary restrictions on the possible event types that can be generated: the elimination and/or softening of these conditions in the object of currrently ongoing research.

Once the reactive scheduling problem is operationally defined and the previous benchmark instances are produced, the way is paved for the next step, consisting in the introduction of a general schedule execution framework, where different proactive and reactive scheduling techniques can be put to the test against different reproducible reactive scheduling problem instances of measurable difficulty.

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

Acknowledgments

Amedeo Cesta, and Riccardo Rasconi's work is partially supported by MIUR (Italian Ministry for Education, University and Research) under the project VINCOLI E PREFERENZE (PRIN). Nicola Policella is currently supported by a Research Fellowship of the European Space Agency, Directorate of Operations and Infrastructure.

References

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

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

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

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

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

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

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

Mc Kay, K. N.; Safayeni, F. R.; and Buzacott, J. A. 1988. Job-Shop Scheduling Theory: What Is Relevant? *Interfaces* 18:84–90.

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

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