Integration of Constraint-Based Off-line and On-line Approaches to Project Scheduling

Riccardo Rasconi*

Institute for Cognitive Science and Technology
Italian National Research Council

riccardo.rasconi@istc.cnr.it

Abstract

Solving a scheduling problem involves considerations of twofold nature: on one hand, the solution must be searched according to specific optimization needs (off-line requirements), while on the other hand, great effort should be employed into producing solutions which can be safely put into execution in unpredictable environments (on-line requirements). In many cases, these two classes of necessities hide mutually conflicting aspects. Aim of my work is to assess the possible integration of off-line and on-line procedures in project scheduling in order to find the best balance between the two, in view of the inherently dynamical utilization of each produced scheduling solution.

Introduction

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

The present work introduces a schedule management schema which tends to integrate the off-line and on-line approaches: according to this schema, the task of the scheduler is not limited to the production of a sequence of activities, as well as the process of controlling schedule executability is not exclusively played on the ground of on-line reaction and activity dispatchment. This work is inspired by the following considerations: (a) regardless the proactive approach employed to produce the baseline schedule, a dynamic analysis on the actual behaviour of the schedule execution is necessary in order to prove, from the operational standpoint, both the efficacy of the choices made and the soundness of the arguments which led to those choices; (b) merely counting on the effectiveness of schedule adjustments at execution time is prone to fostering myopic decisions which may readily result in a complete schedule disruption. Analyzing how the proactive phase may influence (and possibly guide) the reactive phase at execution time is in my opinion as important as assessing the best baseline schedule production strategy on the base of the schedule's particular dynamic behavior. The information that can be extracted from the two phases may reveal mutually useful in order to find an optimal strategy combination, as well as the reasons behind its optimality. For example, one may find an initial solution which is optimal with respect to some criteria of interest (e.g. the makespan), and may wish to know what are the online rescheduling strategies which preserve optimality as much as possible, given some knowledge about the types of exogenous events that may occur during execution.

Aim of the present abstract is to describe the steps I have taken in order to pursue the previous goal. The scheduling problem we specifically focus upon is the project scheduling problem (Brucker *et al.* 1999). These problems are characterized by a rich internal structure. They are based on a network of activities, among which it is possible to identify complex temporal relations that can be used to model a number of variably rigid causal links which normally constrain the tasks in a project. As a further source of complexity, several heterogeneous resources with different capacities serve the activities according to complex modalities.

As a first step, I have implemented an experimental framework which allows to compare different approaches to schedule synthesis and execution in a fair and controlled way. This empirical platform can be used to carry on a set of *reproducible* experiments by (1) simulating the execution of a number of baseline schedules produced with different proactive methods, (2) disturbing their execution with predefined exogenous events, and (3) assessing their behavior by using separate reactive scheduling policies. Different off-line and on-line solving procedures can be compared independently from each other, given the generality of the devised architecture.

Great attention was paid in order to grant measureability and reproducibility of the experimentations. A Reactive Scheduling Problem Benchmark Generator (RSP-BG) has been produced, which, given a project scheduling problem $\mathcal{P}$, returns a number of *exogenous events* (or *disturbs*) that might be fired during the execution of the solution [*baseline($\mathcal{P}$)*]. The production of such events must be strictly related to the structure of each scheduling problem in order to maximize the probability of event acceptance during execution. In order to guarantee experimental fairness, for each initial problem, the RSP-BG allows to produce a specified number of disturbs characterized by a given overall difficulty.

Following the production of a reactive scheduling prob-

*Also affiliated with the Dipartimento di Informatica, Sistemistica e Telematica (DIST) of the University of Genova, Italy

lem benchmarks, a number of explorative experiments have been performed by simulating the execution of baseline schedules synthesized by different proactive techniques and therefore characterized by a different degree of initial "temporal flexibility" (Cheng & Smith 1994).

The Reactive Scheduling Problem Benchmark Generator

Real world uncertainty can be reasonably singled out in the following points: activity delays, growth of activity processing times, lowerings of resource availability, variations in the number of activities, changes in the mutual ordering of the activities.

Currently I have focused our attention on the temporal changes which normally characterize the physical environments, such as delays of the activities start times and/or modifications of activity processing times:

- *delay of the activity start time*: activity a_i undergoes a delay of Δ_{st} time units, at $t = t_{aware}$ ($e_{delay} = \langle a_i, \Delta_{st}, t_{aware} \rangle$);
- *change of activity processing time*: activity a_i's processing time p_i is extended by Δ_p time units, at t_{aware} ($e_p = \langle a_i, \Delta_p, t_{aware} \rangle$).

In order to evaluate the difficulty of each benchmark, proper metrics are introduced 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 μ variation value: $\Delta_\mu = |\mu(\mathcal{P}) - \mu(\mathcal{P}')|$.

One of the possible metrics to use for this purpose is the following, 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):

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

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 (Cesta, Oddi, & Smith 1998), i.e., the ability to use flexibility to absorb temporal variation in the execution of activities: it is a measure of the possibility that a temporal variation concerning an activity is absorbed by the temporal flexibility of the solution instead than generating a deleterious domino effect (the higher the value of $fldt_H$, the less the risk, i.e., the higher the probability of localized changes).

The Schedule Execution Simulation Monitor

The idea behind the open schedule management framework used in this work is simple: an off-line solver produces the baseline solution and delivers it to an on-line module which takes care of assessing its dynamic characteristics by stressing it in a variety of ways.

The overall framework is composed of three modules: the *off-line solver* and the *real world simulator* work off-line and have the job of, respectively, computing the initial solution and generating the exogenous events, intended to disturb the schedule execution; the third module, the *on-line solver*, works on-line and is responsible to complete a simulated execution of the initial solution (the baseline schedule). A number of disturbing events synthesized by the RSP-BG are injected during the simulated execution at the times specified within each event, and their effects are counteracted by the on-line module, which is endowed with a portfolio of *rescheduling* algorithms to the aim of restoring schedule consistency whenever necessary.

Testing Schedule Flexibility

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

Each baseline solution is computed according to different procedures: the result is the production of initial solutions retaining different degree of temporal flexibility. A temporally flexible solution can be described as a network of activities whose start times (and end times) are associated with a set of feasible values (feasibility intervals). Underlying the activity network there exists a Temporal Constraint Network (TCN (Dechter, Meiri, & Pearl 1991)), composed of all the start and end points of each activity (time points), bound to one another through specific values which limit their mutual distances (activity on the arc representation). The search approaches used in our schema focus on decision variables which represent conflicts in the use of the available resources; the solving process proceeds by ordering pairs of activities until all conflicts in the current problem representation are removed. This approach is usually referred to as Precedence Constraint Posting (PCP (Cheng & Smith 1994)), because it revolves around imposing precedence constraints (the *solution constraints*) on the TCN in order to solve the resource conflicts, rather than fixing rigid values to the start times.

In (Cesta, Oddi, & Smith 1998) it is shown that the previous schedule representation inherently provides a certain level of resilience at execution time (i.e. it produces *Flexible Schedules*), even though temporal and resource consistency are guaranteed only if the lower bounds (or upper bounds) from the feasibility intervals are chosen for the time points. In order to overcome the limitation imposed by the flexible schedule, i.e. having only one consistent solution, a generalization of the TCN produced by a PCP phase is proposed in works such as (Cesta, Oddi, & Smith 1998; Policella *et al.* 2004), in which methods for defining a set of both time and resource feasible solutions are presented. This new representation is called *Partial Order Schedule* ($\mathcal{POS}$), a special case of a flexible solution which is obtained by replacing the solution constraints with a new set of constraints that impose a stronger condition on the TCN (*chaining constraints*). A $\mathcal{POS}$ is a flexible solution such that any possible temporal solution is also a resource-consistent assignment. As a third type of solution, fixed time solutions have been tested against Flexible schedules and $\mathcal{POS}$s.

The execution of each baseline schedule has been simulated, in the face of an increasing number of exogenous

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

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

Input: problem P, policies parameter retract and pos
Output: Execution report

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

events, according to the Algorithm 1.

The algorithm is divided in an off-line and an on-line section; in the former, the initial solution can be computed by the `offlineScheduler()` either as a flexible schedule (case FS) or as a $\mathcal{POS}$ (case POS) through the `createPOS()` procedure, depending on the value of the flag `pos`. In the latter, and regardless how the initial solution is produced, it is put into execution according to different modalities, depending on the value of the flag `retract`. At each step of the execution cycle, the environment is sensed for possible disturbs. Afterwards, if `retract` $= true$, the execution algorithm firstly removes all the constraints imposed in the previous solving process (`removeChoice()`), and secondly looks for a new solution (`onlineScheduler()`), possibly creating a new $\mathcal{POS}$. If `retract` $= false$, a new solution is searched leaving the previously imposed solution constraints untouched. In both cases, the algorithm initially checks for temporal consistency after each disturb is acknowledged through the `propagation()` procedure.

Preliminary Experiments and Results

Table 1 shows some preliminary results of our investigation (refer to (Rasconi, Policella, & Cesta 2006) for a more detailed description). As explained earlier, we evaluate the different combinations of off-line/on-line policies — POS-R ($\mathcal{POS}$ + retraction), POS-NR ($\mathcal{POS}$ + no retraction), FS-R (Flexible Schedule + retraction), and FS-NR (Flexible Schedule + no retraction). To make the comparison more complete, we add a further execution mode based on the use of fixed time solution where each activity is assigned a single start time instead of a set of alternatives.

For each entry in the tables, we take into account the following aspects: the number of unexpected events (*number of disturbs*) injected during each single execution, the percentage (with respect to the number of initially solved problems) of the schedules which successfully completed the execution (*% executed*), the execution failure percentage due to the inability to find an alternative solution (*% failed resch.*), the execution failure percentage due to the impossibility to accept the exogenous event on behalf of the TCN (*% refused events*), the average makespan of the solutions at the end of the execution (*mk*), the average difference between the initial and the final makespan (Δ *mk*), the percentage of the performed rescheduling actions with respect to the number of the injected disturbs (*% rescheduling*)[2], the average CPU time, in msecs, to compute the initial solution (*CPU Off-line*), the average CPU time spent to perform all reschedulings during the execution (*CPU On-line*), the *sensitivity* of activity start time w.r.t. the execution process (ψ).

For a fair comparison of the different policies, the data presented in the rightmost part of the table are computed on the basis of the problem instances commonly executed with all the execution strategies.

One of the most striking results that we observe regards the different abilities in preserving the executability of a solution. The outcome shows that the use of partial order schedules tends to lower the success rate in terms of completed executions (*% executed* column). As the table presents, this is mainly due to the dramatic increase in the number of rejected disturbs (*refused events* column). This apparent anomaly can be explained as follows: the creation of a $\mathcal{POS}$ inherently involves a higher level of "constrainedness" in the TCN, in order to guarantee a resource conflict-free solution. This circumstance inevitably makes the TCN more reluctant in accepting new contraints, in the specific case, the constraints which model the exogenous events. Also, note how this effect gets worse as the number of the exogenous events increases (86,87% in the POS-NR case with 1 event, against 56,66% with 5 events).

The rightmost part of Table 1 offers different yet interesting results. One of the most important characteristic to be observed is the extremely low rate of necessary reschedulings exhibited by the POS-R/POS-NR policies (*% resched.* column): this result is all but surprising and confirms the theoretical expectations which motivated the study on the $\mathcal{POS}$. As shown, the need for schedule revision in case of $\mathcal{POS}$ utilization roughly decreases by more than 50% in case of 5 disturbs.

A maybe misleading results is given by the comparison of the final makespan (mk) obtained respectively by using the Retraction and the No-Retraction strategies. In fact, one would expect the R strategies (which allow a greater re-shuffling) to return better makespan values with respect to NR strategies. This is not our case because, as described before, the rescheduling actions are performed by using a less specialized makespan-optimizing procedure (`onlineScheduler` in Algorithm 1) which tends to spoil

[2]We recall that we have a rescheduling action each time the on-line solver is invoked.

	number of disturbs	% executed	failed resch.	refused events	mk	Δmk	% resched.	CPU off-line	CPU on-line	ψ
FS-R		91,04%	2,08%	6,88%	424,60	9,02	24,38%	36242,48	766,15	5,44
POS-R	**1**	87,29%	2,08%	10,63%	419,88	5,07	11,58%	36287,86	303,97	3,00
FS-NR		91,87%	1,25%	6,87%	419,06	3,48	24,14%	36242,48	130,74	1,48
POS-NR		86,87%	2,50%	10,62%	417,11	2,31	11,58%	36287,86	54,59	1,05
fixed time		89,79%	3,75%	6,45%	437,36	21,78	99,75%	36242,48	3035,68	15,16
FS-R		85,21%	3,13%	10,66%	435,54	13,95	23,04%	30259,15	874,70	9,48
POS-R	**2**	76,46%	2,29%	21,25%	429,22	8,41	10,03%	32371,82	674,86	5,17
FS-NR		85,62%	2,71%	11,66%	427,90	6,30	22,88%	30259,15	258,50	3,07
POS-NR		73,95%	5,00%	21,04%	424,74	3,93	9,56%	32371,82	97,46	2,02
fixed time		81,25%	8,54%	10,20%	446,62	25,02	99,53%	30259,15	2768,37	17,73
FS-R		79,17%	3,96%	16,87%	449,84	19,40	20,27%	26318,37	963,16	11,83
POS-R	**3**	69,58%	2,92%	27,50%	441,49	11,93	9,41%	28406,98	675,18	6,96
FS-NR		80,00%	2,50%	17,50%	439,66	9,23	22,37%	26318,37	371,03	4,08
POS-NR		67,71%	5,41%	26,87%	436,24	6,68	9,63%	28406,98	151,26	3,16
fixed time		77,50%	6,87%	15,62%	458,32	27,89	99,22%	26318,37	3716,15	19,77
FS-R		70,21%	4,17%	25,63%	464,12	28,85	22,45%	25682,40	2391,92	17,36
POS-R	**5**	60,42%	3,13%	36,46%	455,56	21,07	10,57%	27544,98	1748,56	13,12
FS-NR		70,41%	3,33%	26,25%	447,42	12,15	21,66%	25682,40	646,90	5,80
POS-NR		56,66%	7,08%	36,25%	444,68	10,18	10,48%	27544,98	289,17	4,81
fixed time		67,08%	8,33%	24,58%	465,56	30,29	98,43%	25682,40	6721,48	19,74

Table 1: Summarizing data for each execution strategy (the values in the last six columns are computed on the intersection set of all successfully executed j100 problems)

the makespan quality. On the other hand, the NR strategy that tries to maintain the schedule continuity is also able to obtain a preservation of makespan values.

Another interesting aspect can be observed by comparing the *CPU on-line* values between the *Retraction* and *No Retraction* strategies. In general, the Retraction methods require a higher CPU on-line load because the removal of the solution constraints inevitably re-introduces some resource conflicts that must be solved by rescheduling. But the intriguing result lies in the fact that this difference in the CPU on-line rates stands *despite the comparable amount of performed reschedulings*. Let us look at the difference between the FS-R and FS-NR rates: it can be seen that, in the 5 events case, we have 2392 ms. (FS-R) against 647 ms. (FS-NR), although the number of performed reschedulings is practically the same ($\approx 21\%$)! The same effect can be observed between the POS-R and POS-NR cases: 1748 ms. against 289 ms, notwithstanding the same ($\approx 10.5\%$) number of reschedulings. This circumstance can be explained as follows: NR execution modes retain all the temporal constraints of the previous solution: hence, the rescheduler is bound to work on a smaller search space, finding the next solution almost immediately.

Ongoing Work

The analysis being performed on the considered scheduling benchmarks is returning several interesting information. Some results confirm the expectations while other require a certain level of analysis in order to be correctly understood. For instance, the rigid behavior exhibited by the fixed time schedules when confronted with dynamically variable environments is totally confirmed, as confirmed is the behavior of schedules characterized by a more flexible nature.

However, among several other aspects, the experiments revealed a scarce capability in accepting exogenous events on behalf of the $\mathcal{POS}$: post-experimental analysis about this unexpected effect has shown that this drawback is due to an increase of constrainedness in the TCN, necessary to guarantee a resource conflict-free solution at all times. This has suggested several research lines which are the object of ongoing work, such as the production of a different class of $\mathcal{POS}$s, through the development of alternative chaining procedures aimed at minimizing the inevitable constrainedness increase in the TCN. Moreover, the observed dynamic behavior of the schedules suggests to study the introduction of different reactive techniques. For instance, a possible approach under current development is based on *informed retraction* procedures, where the constraint removal strategy is preceded by a search phase to determine the constraints which are to be retracted, depending on the particular dynamic requirements.

Acknowledgments. I would like to thank Amedeo Cesta and Nicola Policella for their continuous and invaluable support in this research, as well as all my colleagues at ISTC-CNR.

References

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

Cesta, A.; Oddi, A.; and Smith, S. F. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings, AIPS-98*, 214–223. AAAI Press.

Cheng, C., and Smith, S. F. 1994. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings, AAAI-94*, 1086–1091. AAAI Press.

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

Policella, N.; Oddi, A.; Smith, S. F.; and Cesta, A. 2004. Generating Robust Partial Order Schedules. In *CP 2004*, volume 3258.

Rasconi, R.; Policella, N.; and Cesta, A. 2006. Fix the schedule or solve again: Comparing constraint-based approaches to schedule execution. In *Proceedings of Constraint Satisfaction Techniques for Planning and Scheduling Problems Workshop ICAPS 2006*. To appear.