From Precedence Constraint Posting to Partial Order Schedules

A CSP approach to Robust Scheduling

Nicola Policella [a], Amedeo Cesta [a], Angelo Oddi [a], and Stephen F. Smith [b]
[a] *ISTC-CNR*
Institute for Cognitive Science and Technology, National Research Council of Italy
E-mail: name.surname@istc.cnr.it
[b] *The Robotics Institute, Carnegie Mellon University*
E-mail: sfs@cs.cmu.edu

Constraint-based approaches to scheduling have typically formulated the problem as one of finding a consistent assignment of start times for each goal activity. In contrast, we are pursuing an approach that operates with a problem formulation more akin to least-commitment frameworks, where the objective is to post sufficient additional precedence constraints between pairs of activities contending for the same resources to ensure feasibility with respect to time and resource constraints. One noteworthy characteristic of this Precedence Constraint Posting (PCP) approach, is that solutions generated in this way generally encapsulate a set of feasible schedules (i.e., a solution contains the sets of activity start times that remain consistent with posted sequencing constraints). Such solutions can offer advantages when there is temporal uncertainty associated with executing activities.

In this paper, we consider the problem of generating temporally flexible schedules that possess good robustness properties. We first introduce the concept of a Partial Order Schedule ($\mathcal{POS}$), a type of temporally flexible schedule in which each embedded temporal solution is also guaranteed to be resource feasible, as a target class of solutions that exploit flexibility in a robust way. We then present and analyze two PCP-based methods for generating $\mathcal{POS}$s. The first method uses a pure least commitment approach, where the set of all possible time-feasible schedules is successively winnowed into a smaller resource-feasible set. The second method alternatively utilizes a focused analysis of one possible solution, and first generates a single, resource-feasible, fixed-times schedule. This point solution is then transformed into a $\mathcal{POS}$ in a second post processing phase. Somewhat surprisingly, this second method is found to be a quite effective means of generating robust schedules.

Keywords: Scheduling, Robustness, Constraint Programming.

1. Introduction

In most practical scheduling environments, off-line solutions can have a very limited lifetime and scheduling has to consider the on-line process of responding to unexpected and evolving circumstances. In such environments, insurance of robust response is generally the first concern. Unfortunately, the lack of guidance that might be provided by a schedule often leads to myopic, sub-optimal decision-making.

One way to address this problem is *reactively*, through schedule repair. To keep pace with execution, the repair process must be fast, so that the execution of the schedule can be re-started as soon as possible. To be maximally effective, a repair should also be complete in the sense of accounting for all changes that have occurred, while attempting to avoid the introduction of new changes. As these two goals can be conflicting, a compromise solution is often required. Different approaches exist and they tend to favor either timeliness [33] or completeness [16] of the reactive response.

An alternative, *proactive* approach to managing execution in dynamic environments is to focus on building schedules that retain flexibility and are able to absorb some amount of unexpected events without rescheduling. One technique consists of factoring time and/or resource redundancy into the schedule, taking into account the types and nature of uncertainty present in the target domain [11]. Alternatively, it is sometimes possible to construct an explicit set of contingencies (i.e., a set of complementary solutions) and use the most suitable with respect to the actual evolution of the envi-

AI Communications
ISSN 0921-7126, IOS Press.

ronment [15]. Both of these proactive techniques presume an awareness of the possible events that can occur in the operating environment, and in some cases, these knowledge requirements can present a barrier to their use.

In this paper, we consider a less knowledge intensive approach to generating robust schedules: to simply build schedules that retain flexibility where problem constraints allow. We take two solution properties – the flexibility to absorb unexpected events and a solution structure that promotes localized change – as our primary solution robustness objectives, to promote both high reactivity and solution stability as execution proceeds.

We develop and analyze two methods for producing temporally flexible schedules. Both methods follow a general Precedence Constraint Posting (PCP) scheduling strategy, which aims at the construction of a partially ordered solution, and proceeds by iteratively introducing sequencing constraints between pairs of activities that are competing for the same resources. The two methods considered differ in the way that they detect and analyze potential resource conflicts. The first method uses a pure least commitment approach. It computes upper and lower bounds on resource usage across all possible executions according to the exact computations proposed in [25] (referred to as the resource envelope), and successively winnows the total set of time feasible solutions into a smaller resource-feasible set. The second method, alternatively, takes the opposite extreme approach. It utilizes a focused analysis of one possible execution (the early start time profile) as in [6,8], and establishes resource feasibility for a specific single-point solution (the early start time solution). This second approach is coupled with a post-processing phase which then transforms this initially generated point solution into a temporally flexible schedule.

The paper is organized as it follows. We first review basic concepts of constraint-based scheduling and the particular approach that underlies our work: Precedence Constraint Posting. We then introduce the notion of Partial Order Schedules as a target class of solutions that exploit temporal flexibility in a robust way. Next we describe the two PCP-based scheduling methods mentioned above. These are evaluated on a challenging benchmark from the Operations Research (OR) literature, and the solution sets produced in each case are compared with respect to solution robustness properties. Finally, we draw some conclusions about the proposed solving methods.

A preliminary version of this work appeared in [29]. The current paper extends this work in several ways. First, it emphasizes the key role of constraint programming in formulating precedence constraint posting methods and in generating $\mathcal{POS}$s. Second, the paper includes a more comprehensive experimental analysis using a larger size benchmark problem set. Finally, the paper extrapolates from the results of the comparison, and summarizes the broader characteristics of the two-step *Solve & Robustify* model for generating robust schedules.

2. Constraint-based Scheduling

Our approach is characterized by the use of the Constraint Satisfaction Problem paradigm (CSP): a CSP consists of a network of constraints defined over a set of variables where a solution is an assignment to the variables that satisfies all constraints.

Constraints do not simply represent the problem but also play an important role in the solving process by effectively narrowing the space of possible solutions. Constraint satisfaction and propagation rules have been successfully used to model, solve and reason about many classes of problems in such diverse areas as scheduling, temporal reasoning, resource allocation, network optimization and graphical interfaces. In particular, CSP approaches have proven to be an effective way to model and solve complex scheduling problems (see for instance [17,32,33,5,3,8]). The use of variables and constraints provides representational flexibility and reasoning power. For example, variables can represent the start and the end times of an activity, and these variables can be constrained in arbitrary ways.

In the remainder of this section we first provide a formal definition of Constraint Satisfaction Problem and then an overview of the different ways in which this paradigm has been used to solve scheduling problems.

2.1. Constraint Satisfaction Problem

A *constraint satisfaction problem*, CSP, consists of a finite set of variables, each associated with a domain of values, and a set of constraints that define the relation between the values that the variables can assume. Therefore a CSP is defined by the tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ where:

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is a set of n variables;

- $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ is the set of corresponding domains for any variable, that is, $v_1 \in D_1$, $v_2 \in D_2$ and $v_n \in D_n$;
- $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, is a set of m constraints, $c_k(v_1, v_2, \ldots, v_n)$, that are predicates defined on the Cartesian product of the variable domains, $D_1 \times D_2 \times \ldots \times D_n$.

A *solution* is a value assignment to each variable, from its domain,

$$(\lambda_1, \lambda_2, \ldots, \lambda_n) \in D_1 \times D_2 \times \ldots \times D_n$$

such that the set of constraints is satisfied. A fundamental aspect of constraint satisfaction problems is that a CSP instance $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ can be conceptualized as a constraint graph, $G = \{V, E\}$. For every variable $v_i \in \mathcal{V}$, there is a corresponding node in V. For every set of variables connected by a constraint $c_j \in \mathcal{C}$, there is a corresponding hyper-edge in E. In the particular case in which only binary constraints (each constraint involves at most two variables) are defined the hyper-edges become simple edges. In the following sections we consider a particular type of binary CSP: the Simple Temporal Problem, or STP, [12].

Constraint Programming is an approach to solving combinatorial optimization problems based on the CSP representation [24,21,35]. This framework is based on the combination of sophisticated search technologies and constraint propagation. Constraint propagation consists of using constraints actively to prune the search space. These propagation techniques are generally polynomial w.r.t. the size of the problem and aim at reducing the domains of variables involved in the constraints by removing the values that cannot be part of any feasible solution. Different techniques with different pruning power have been defined for different types of constraint.

As described in [13], "in general, constraint satisfaction tasks, like finding one or all solutions or the best solution, are computationally intractable, NP-hard". For this reason a polynomial constraint propagation process cannot be complete, that is, some infeasible values may still sit in the domains of the variables and thus decisions are necessary to find a complete feasible valuation of the variables.

2.2. CSP approaches to Scheduling Problems

Scheduling problems are very hard problems: for instance simple scheduling problems like job-shop scheduling are NP-hard [18]. Therefore, scheduling represents an important application for constraint directed search, requiring the definition of heuristic commitments and propagation techniques. A first attempt to model scheduling problems like CSP instances is given in [31] where the authors describe a graph based formalism to represent a job-shop scheduling problem.

Different constraint programming approaches have been developed in this direction, for instance, the reader can refer to [3] for a thorough analysis of different constraint based techniques for scheduling problems. The work of Constraint directed Scheduling of the 80's (see for example [17,32,33]) has developed into Constraint-based Scheduling approaches in the late 90's (see [2,27,5]). These approaches are based on the representation of a scheduling problem and the search for a solution to it by focusing upon the constraints in the problem. The search for a solution to a CSP can be viewed as modifying the constraint graph $G = \{V, E\}$ by addition and removal of constraints, where the constraint graph is an evolving representation of the search state, and a solution is a state with a single value remaining in the domain of each variable, and all constraints are satisfied.

Research in constraint-based scheduling (e.g., [32, 26]) has typically formulated the problem as that of finding a consistent assignment of start times for each goal activity. Under this model, decision variables are time points that designate the start times of various activities and CSP search focuses on determinating a consistent assignment of start time values. In contrast, we are investigating approaches to scheduling that operate with a problem formulation more akin to least-commitment frameworks. In this formulation, referred to as *Precedence Constraint Posting* [34], the goal is to post sufficient additional precedence constraints between pairs of activities for the purpose of pruning all inconsistent allocations of resources to activities. The following section is dedicated to discussion of these approaches.

3. Precedence Constraint Posting

In this paper we follow a different research trend with respect to solving scheduling problems, that is based on the general concept of "temporal flexibility". This approach, introduced in [34,10] for problems with binary resources and then extended to more general problems in an amount of later work, is based on the fact that the relevant events on a scheduling problem can be represented in a temporal CSP, usually called Simple Temporal Problem (STP) [12]. The

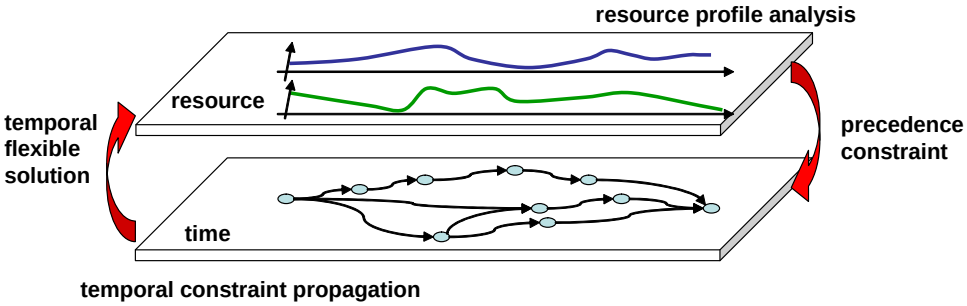


Fig. 1. Precedence Constraint Posting Schema

search schema used in this approach focuses on decision variables which represent conflicts in the use of the available resources; the solving process proceeds by ordering pair of activities until all the current resource violations are removed. This approach is usually referred to as the Precedence Constraint Posting, PCP, because it revolves around imposing precedence constraints to solve the resource conflicts, rather than fixing rigid values to the start times.

The general schema of these approaches is provided in Fig. 1. The approach consists of representing, analyzing, and solving different aspects of the problem in two separate layers. In the former the temporal aspects of the scheduling problem like activity durations, constraints between pairs of activities, due dates, release time, etc., are considered. The second layer, instead, represents and analyzes the resource aspects of the problem.[1] Let us now explain the details of the two layers.

Time layer. The temporal aspects of the scheduling problems are represented through an STP (simple temporal problem) network [12]. This is a temporal graph in which the set of nodes represents a set of temporal variables named time points, tp_i, while temporal constraints, of the form $tp_i - tp_j \leq d_{ij}$, define the distances among them. Each time point has initially a domain of possible values equal to $[0, H]$ where H is the horizon of the problem (generally H can be infinite).

The problem is represented by associating with each activity a pair of time points which represent, respectively, the start and the end time of the activity. A temporal constraint between two time-points may define either ordering constraints between two activities (when the two time-points do not belong to the same activity) or activity durations (when the two time-points belong to the same activity).

[1] A similar distinction between temporal and resource aspects of the scheduling problem is introduced in [16].

By propagating the temporal constraints it is possible to bound the domains of each time point, $tp_i \in [lb_i, ub_i]$. In the case of empty domains for one or more time points the temporal graph does not admit any solution. In [12] it has been proved that it is possible to completely propagate the whole set of temporal constraints in polynomial time, $O(n^3)$, and, moreover, a solution can be obtained selecting for each time point its lower bound value, $tp_i = lb_i$ (this solution is named the *earliest start-time solution*).

The temporal layer then, considering the temporal aspects of a scheduling problem, provides, in polynomial time (using constraint propagation) a set of solutions defined by a temporal graph. This result is taken as input in the second layer. In fact, at this stage we have a set of temporal solutions (time feasible) that need to also be proven to be resource feasible.

Resource layer. This layer takes into account the other aspect of the scheduling problem, namely resources. The problem is that there are constraints on resource utilization (i.e., capacity). Resources can be single or multi capacitive, and reusable or consumable.

The input of this layer is a temporally flexible solution – a set of temporal solutions (see also Fig. 1). Like in the previous layer it is possible to use constraint propagation (i.e. resource propagation) to reduce the search space. Even though there are different methodologies described in the literature, see [27,22], these are not sufficient in general. In fact these are not complete, that is, they are not able to prune all inconsistent temporal solutions. Therefore, it is necessary to introduce a method for deciding among the possible alternatives.

For this reason a PCP procedure uses a *Resource Profile* to analyze resource usage over time and detect periods of resource violations and the set of activities, or *contention peaks*, which create this situation. The procedure then proceeds to post additional

constraints on the time layer to level (or solve) one or more detected contention peaks. These new constraints are propagated in the time layer to check the temporal consistency. Then the time layer provides a new temporally flexible solution that is analyzed again using the resource profiles. The search stops when either the temporal graph becomes inconsistent or the resource profiles are consistent with the resource capacities.

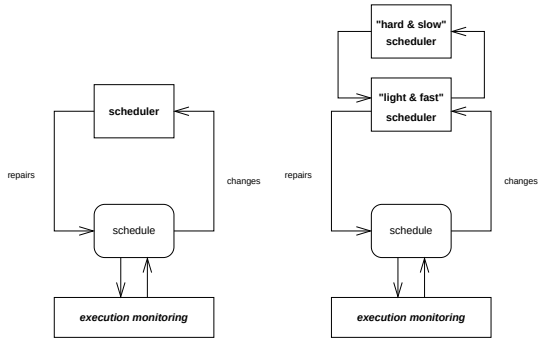
Remarks. The outcome of a PCP solver is a STP that not only contains the temporal constraints belonging to the initial problem, but also the additional precedences which have been added during the resolution process. The goal of PCP approaches is to retain the temporal flexibility of the underlying temporal network as much as possible (somehow maximizing the domain size of the time points).

This kind of flexible solutions can offer advantages when there is uncertainty associated with executing activities. Unfortunately, exploiting this flexibility might often lead to other kinds of inconsistencies. This is due to the fact that not all allocations in time allowed by the temporal network are also resource consistent, and there might be many value assignments to time points which, though temporally consistent, could trigger resource conflict in the solution. For this reason, in the next section we introduce a solution paradigm to go beyond flexible schedules.

4. Flexible Solutions, Robustness, and Partial Order Schedules

In any given scheduling domain, there can be different sources of executional uncertainty: durations may not be exactly known, there may be less resource capacity than expected (e.g., in a factory context, due to a breakdown of a sub-set of the available machines), or new tasks may need to be taken into account pursuant to new requests and/or goals. Current research approaches are based on different meanings of robust solutions, e.g., the ability of preserving solution qualities and/or solution stability. The concept of robustness pursued in this work can be viewed as execution-oriented; a solution to a scheduling problem will be considered *robust* if it provides two general features: (1) the ability to absorb exogenous and/or unforeseen events without loss of consistency, and (2) the ability to keep the pace with the execution guaranteeing a prompt answer to the various events.

We base our approach on the generation of flexible schedules, i.e., schedules that retain temporal flexibility. We expect a flexible schedule to be easy to change, and the intuition is that the degree of flexibility in this schedule is indicative of its robustness. Figure 2 underlines the importance of having a flexible solution. In Fig. 2(a) a schedule is given to an executor (it can be either a machine or a human) that manages the different activities. If something happens (i.e., an unforeseen event occurs) the executor will give feedback to a scheduler module asking for a new solution. Then, once a new solution is computed, it is given back to the executor. In Fig. 2(b), instead, the execution of a flexible schedule is highlighted. The substantial difference in this case is that the use of flexible solutions allows the introduction of two separate phases of rescheduling: the first consists of facing the change by immediate means like its propagation over the set of activities. In practice, in this phase the flexibility characteristics of the solution are exploited (for this reason we named this module *light & fast scheduler*). Of course it is not always possible to face an unforeseen event by using only "light" adjustments. In this case, it will be necessary to ask for a more complete scheduling phase. This will involve a greater number of operations than in the light phase. This module has been named *hard & slow scheduling*. It is worth noting that the use of flexible schedules makes it possible to bypass the last, more complicated, phase in favor of a prompt answer.[2] Depending on the scheduling problem considered we can have very different behaviors of the two scheduler modules: for instance, in the next sections, we discuss the RCPSP/max problem. This problem implies an exponential complexity for the hard & slow module while



(a) General rescheduling phase (b) Rescheduling phase using a flexible solution

Fig. 2. Rescheduling actions during the execution

[2] However the reader should note that these solutions are in general sub-optimal.

for the light & fast module a polynomial approach is sufficient.

Our approach adopts a graph formulation of the scheduling problem and focuses on generation of *Partial Order Schedules* ($\mathcal{POS}$s).

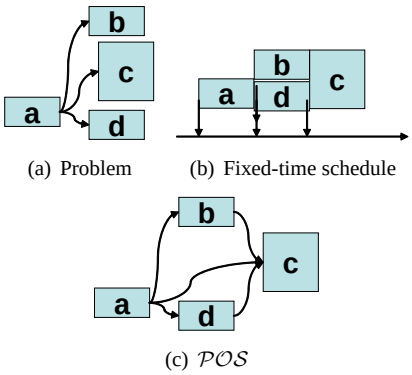
Definition 1 (Partial Order Schedule) *A Partial Order Schedule $\mathcal{POS}$ for a problem $\mathcal{P}$ is an activity network, such that any possible temporal solution is also a resource-consistent assignment.*

Within a $\mathcal{POS}$, each activity retains a set of feasible start times, and these options provide a basis for responding to unexpected disruptions.

An attractive property of a $\mathcal{POS}$ is that reactive response to many external changes can be accomplished via simple propagation in an underlying temporal network (a polynomial time calculation); only when an external change exhausts all options for an activity is necessary to recompute a new schedule from scratch. In fact the augmented duration of an activity, as well as a greater release time, can be modeled as a new temporal constraint to post on the graph. To propagate all the effects of the new edge over the entire graph it is necessary to achieve the *arc-consistency* of the graph (that is, ensure that any activity has a legal allocation with respect the temporal constraints of the problem).

It is worth noting that, even though the propagation process does not consider consistency with respect resource changes, it is guaranteed to obtain a feasible solution by definition. Therefore a partial order schedule provides a means to find a new solution and ensures that it can be computed in a fast way.

Example 1 *Figure 3 summarizes the aspects introduced so far. We have a problem, Fig. 3(a), with four activities $\{a,b,c,d\}$ which require respectively $\{1,1,2,1\}$ resource units during their execution (note the double height of activity c, representing the higher resource demand). The activity network describes precedence constraints $\{a \prec b, a \prec c, a \prec d\}$. Moreover, supposing that the only available resource has maximum capacity equal to 2, we have a resource conflict among the activities $\{b,c,d\}$. For this problem we consider two alternative solutions: a fixed-time solution and a partial order schedule, Fig. 3(c). While the fixed-time solution consists of a complete activity allocation the other one consists of a further ordering of the subset of activities $\{b,c,d\}$: $\{b \prec c, d \prec c\}$. For the fixed-time schedule it is the case that any perturbation creates a conflict and requires a rescheduling phase. On the other hand, the $\mathcal{POS}$ is able to preserve the consistency of the solution thanks to the additional constraints: delay of any of the activities does not require rescheduling to reinforce solution consistency.*



(a) Problem (b) Fixed-time schedule

(c) $\mathcal{POS}$

Fig. 3. Types of scheduling solutions

Before concluding we make a further remark about partial order schedules. In [31] the authors introduce the disjunctive graph representation of the classical job shop scheduling problem and describe how a solution can be achieved by solving all the disjunctive constraints and transforming each into a conjunctive one. Also in our case, solving all disjunctive constraints is required to achieve a $\mathcal{POS}$. Now, the disjunctive graph representation can be extended to the more general case where multi-capacity resources are defined. In this case "disjunctive" hyper-constraints among activities that use the same resource are introduced. Based on this representation we can note that a partial order schedule is obtained once any disjunctive hyper-constraint is solved. In this case, a set of precedence constraints is posted to solve each hyper-constraint.

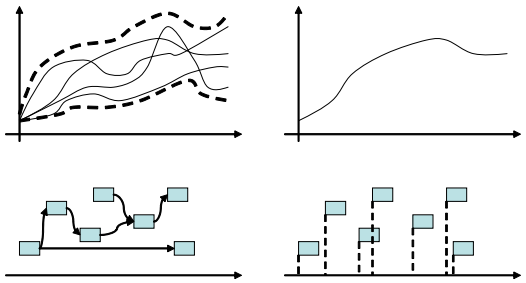
5. Partial Order Schedules: a constraint-based approach

The ability to manage precedence constraints during the solving process represents the main reason for which we have pursued a PCP approach. In fact, solutions generated in this way generally represent a set of feasible schedules (i.e., the sets of activity start times that remain consistent with posted sequencing constraints), as opposed to a single assignment of start times. In the following sections we will describe several algorithms, in which $\mathcal{POS}$s are produced by adding new precedence constraints.

The remainder of this section is dedicated to describe the basic framework and the heuristics that are

used for the analysis of the resource profile and the synthesis of new constraints. However, one important issue that needs to be explored is how to compute these profiles. In fact, the input temporal graph represents a set of solutions, possibly infinite, and to consider all the possible combinations is impossible in practice. A possible affordable alternative consists of computing bounds for the resource utilization. Examples of bounds can be found in [14,9,22,25]. It is worth noting that considering resource bounds assures generation of a partial order schedule at the end of the search process. In fact, the search can stop when the resource profile is consistent with respect to the capacity constraints; this implies that all temporal solutions represented by the temporal graph are also resource feasible.

A different approach to dealing with resources consists of focusing attention on a specific temporal solution and its resource utilization. Even though the precedence constraint posting method produces a temporal graph when the resource utilization is consistent with resource capacity, this graph, in general, is not a $\mathcal{POS}$. Indeed, this process only assures that the final graph contains at least one resource feasible solution (the one for which the resource utilization is considered); some of the temporal solutions may not be resource feasible. Thus it is necessary to add a method which is capable of transforming the resulting temporal graph into a partial order schedule.



(a) Bounds of the resource utilization for the set of solutions defined by a temporal graph

(b) Resource utilization of a single temporal solution

Fig. 4. Two different ways to consider the resource utilization

Figure 4 summarizes the two alternative resource profiles. In the first case resource bounds are used to consider all the temporal solutions and their associated resource utilization (Fig. 4(a)). Alternatively, only one temporal solution of the set is considered in the second case (Fig. 4(b)). This allows reasoning about one precise resource profile but, in fact, produces temporal graphs that are not in general $\mathcal{POS}$s.

We proceed now by introducing the reference scheduling problem (RCPSP/max) and then the core components that need to be explained to complete the description of the approach. In Section 6 we will discuss how different ways of computing and using resource profiles lead to different PCP-style algorithms.

5.1. *The Reference Scheduling Problem:* RCPSP/max

We adopt the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max, as a reference problem [4]. The basic entities of interest in this problem are *activities*. The set of activities is denoted by $A = \{a_1, a_2, \ldots a_n\}$ where each activity has a fixed *processing time*, or *duration*, p_i and must be scheduled without preemption.

A *schedule* is an assignment of start times to activities $a_1, a_2, \ldots a_n$, i.e. a vector $S = (s_1, s_2, \ldots, s_n)$ where s_i denotes the start time of activity a_i. The time at which activity a_i has been completely processed is called its *completion time* and is denoted by e_i. Since we assume that processing times are deterministic and preemption is not permitted, completion times are determined by:

$$e_i = s_i + p_i \tag{1}$$

Schedules are subject to two types of constraints, *temporal constraints* 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,

$$l_{ij}^{min} \leq s_j - s_i \leq l_{ij}^{max} \tag{2}$$

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 $S = (s_1, s_2, \ldots, s_n)$ is *time feasible*, if all inequalities given by the activity precedences/time lags 2 and durations 1 hold for start times 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. they are released when no longer required by an activity and are then available for use by another activity. Each activity a_i requires of the use of req_{ik} units of the resource r_k during its processing time p_i. Each resource r_k has a limited capacity of cap_k units.

A schedule S is *resource feasible* if at each time t the demand for each resource $r_k \in R$ does not exceed its capacity cap_k, i.e.

$$Q_k^S(t) = \sum_{s_i \leq t < e_i} req_{ik} \leq cap_k. \tag{3}$$

A schedule S is called *feasible* if it is both time and resource feasible.

The RCPSP/max problem is a very complex scheduling problem: in fact not only the optimization version but also the feasibility problem is NP-hard [4]. The reason for this NP-hardness result lies in the presence of maximum time-lags. In fact these constraints imply the presence of deadline constraints, transforming feasibility problems for precedence-constrained scheduling to scheduling problems with time windows.

It is worth remarking that the start times and the end times of each activity correspond to the time points of the time-layer described above. Moreover each time point will have an associated resource production/consumption ru: given an activity a_i and the resource r_k, the start time has a resource production $ru = req_{ik}$ whereas the end time has a consumption $ru = -req_{ik}$.

It is also worth noticing that when in the following we say that a precedence constraint is added between two activities, $a_i \prec a_j$, it means that a precedence constraint is added between the end time of a_i and the start time of a_j, i.e., $s_j \geq e_i$.

5.2. The Core Constraint-based Framework

The core of the implemented framework is based on the greedy procedure described in Algorithm 1. Within this framework, a solution is generated by progressively detecting time periods where resource demand is higher than resource capacity and posting sequencing constraints between competing activities to reduce demand and eliminate capacity conflicts. As explained above, after the current situation is initialized with the input problem, $S_0 \leftarrow \mathcal{P}$, the procedure builds an estimate of the required resource profile according to the current temporal precedences in the network. This analysis can highlight contention peaks, where resource needs are greater than resource availability. If there are resource violations new constraints are synthesized and posted on the current situation. The search proceeds until either the temporal graph becomes inconsistent or a solution is found.

5.2.1. How to identify conflicts

The starting point in identifying the possible conflicts in a situation is to compute the possible contention peaks. A couple of definitions are necessary before proceeding:

Algorithm 1 GREEDYPCP($\mathcal{P}$)

Input: a problem $\mathcal{P}$
Output: a solution S (or the empty set otherwise)

```
S_0 ← P
if Exists an unresolvable conflict in S_0 then
  S ← ∅
else
  C_s ← CONFLICTSET(S_0)
  if C_s = ∅ then
    S ← S_0
  else
    {a_i ≺ a_j} ← LEVELINGCONSTRAINT(C_s)
    S_0 ← S_0 ∪ {a_i ≺ a_j}
    S ← GREEDYPCP(S_0)
return S
```

1. a *contention peak* is a set of activities whose simultaneous execution exceeds the resource capacity. A contention peak designates a conflict of a certain size (corresponding to the number of activities in the peak);
2. a *conflict* is a pair of activities $\langle a_i, a_j \rangle$ belonging to the same contention peak.

In Algorithm 1 the function CONFLICTSET(S_0) collects all peaks in the current schedule, ranks them, picks the most critical one and then selects a conflict from this last peak. The conflict is solved by ordering the conflicting activities with a new precedence constraint, $a_i \prec a_j$. In the remainder of this section we describe first how conflicts can be extracted from contention peaks and, second, how to select and solve the more critical conflict. The discussion of how to to identify contention peaks is deferred until Section 6.

A first way to extract a conflict from a peak is by *pairwise selection* [34]. This consists of collecting any competing pair of activities associated with each peak. The myopic consideration on any pair of activities in a peak can, however, lead to an over commitment. For example, consider a resource r_k with capacity $cap_k = 4$ and three activities a_1, a_2 and a_3 competing for this resource. Assume that each activity requires respectively 1, 2 and 3 units of the resource. Taking into account all possible pairs of activities will lead to consideration of the pair $\langle a_1, a_2 \rangle$. But the sequencing of this pair will not resolve the conflict because the combined capacity requirement does not exceed the capacity.

An enhanced conflict selection procedure which can avoid this problem is based on the identification of *Minimal Critical Sets* [23] inside each contention peak. A *Minimal Critical Set*, MCS, is *a contention peak with*

the property that no proper subset of activities contained in the MCS *is itself a conflict*. The important advantage of isolating MCSs is that a single precedence relation between any pair of activities (or conflict) in the MCS eliminates the resource conflict. Let us consider again the previous example: in this case the only MCS is $\{a_2, a_3\}$ and both the precedence constraints $a_2 \prec a_3$ and $a_3 \prec a_2$ level the peak. Application of this method can be seen generally as a filtering step. Indeed, it extracts from each contention peak those subsets that are necessary to solve.

MCS analysis has been used in [23], where MCSs are seen as particular cliques that are collected via systematic search of an activity "intersection graph" (which is constructed starting from the temporal information). The unfortunate drawback of this approach is the exponential nature of the intersection graph search, which prohibits the use of this basic approach on scheduling problems of any interesting size. In [7], it is shown that much of the advantage of this type of global conflict analysis can be retained by using an approximate procedure for computing MCSs. In particular the authors define two polynomial strategies for sampling MCSs from a peak of size p. Both strategies first sort the activities in each peak according to their resource usage (largest first), then they collect the MCSs by visiting such a list. The two methods are named *linear* and *quadratic* according to their complexity (they respectively collect $O(p)$ and $O(p^2)$ elements). In what follows we utilize three different operators for gathering conflicts: the simple *pairwise selection*, and the increasingly accurate *linear* and *quadratic* MCS sampling strategy.

5.2.2. Select and solve conflicts

The basic idea is to resolve the conflict that is the most "dangerous" and solve it with a commitment as small as possible. More specifically, the following heuristics are assumed:

Ranking conflicts: for evaluating MCSs we have used the heuristic estimator $K()$ described in [23]. The heuristic estimator $K()$ chooses the MCS with highest value. A conflict is unsolvable if no pair of activities in the conflict can be ordered. Basically, $K()$ will measure how close a given conflict is to being unsolvable.

Slack-based conflict resolution: to choose an order between the selected pair of activities we apply *dominance conditions* that analyze the reciprocal flexibility between activities [34]. In the case where both orderings are feasible, the choice which retains the most temporal slack is taken.

These two heuristics have been selected because they adopt a minimal commitment strategy with respect to preserving temporal slack, and this again favors temporal flexibility.

6. Two Profile-Based Solution Methods

As suggested previously, different solution methods can be specified by varying the approach taken to generation and use of resource profiles. Here, we consider two extreme approaches: (1) a pure least commitment approach, which uses the resource envelope computation introduced in [25] to anticipate all possible resource conflicts and establish ordering constraints on this basis, and (2) an "inside-out" approach which uses the focused analysis of early start time profiles introduced in [6] to first establish a resource-feasible early start time solution and then applies a chaining procedure to expand this early start time solution into a $\mathcal{POS}$.

Figure 5 gives a sketched view of both approaches to building $\mathcal{POS}$s investigated in this paper with respect to the underlying search space. The different sets represent temporal solutions associated to a graph obtainable by posting new constraints on the initial scheduling problem. The use of the resource envelope (on the

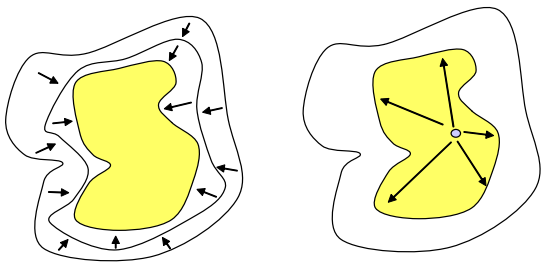


Fig. 5. Two Profile-Based Solution Approaches

left hand picture) implies an iterative reduction of the space of all the possible temporal solutions until this set contains only feasible solutions. On the contrary, the inside-out procedure first computes a single solution (a point in the search space) then generalizes the result to obtain a set of solutions (see the right hand picture in Fig. 5). Thus on one hand the envelope-based approach considers all temporal solutions at each stage and tries to select decisions which reduce as minimally as possible this set (i.e. least commitment). Conversely, in the two step procedure the final objective, i.e. to obtain a partial order schedule, is neglected in the first

step where the actual goal is to find a feasible fixed-time solution. Only in the successive phase the aim of building flexible solutions is taken under consideration.

6.1. Least-Commitment Generation Using Envelopes

The perspective of a "pure" least commitment approach to scheduling consists of carrying out a refinement search that incrementally restricts a partial solution (the possible temporal solutions $\tau \in S_T$) with resource conflicts until a set of solutions (a $\mathcal{POS}$ in our case) is identified that is resource consistent. A useful technical result has been introduced in [25]. This potentially can contribute to the effectiveness of this type of approach with an exact computation of the so-called Resource Envelope. According to the terminology introduced previously we can define the Resource Envelope as follows:

Definition 2 (Resource Envelope) *Let S_T be the set of temporal solutions and let τ be in S_T. For each resource r_k we define the Resource Envelope in terms of two functions:*

$$L_k^{max}(t) = \max_{\tau \in S_T} \{Q_k^\tau(t)\}$$

$$L_k^{min}(t) = \min_{\tau \in S_T} \{Q_k^\tau(t)\}$$

By definition the Resource Envelope represents the *tightest possible resource-level bound for a flexible plan*.

A brief introduction of the resource envelope as described in [25] requires, given a time instant t, the following partition of the set of time points (or events):

- B_t: the set of events tp_i which must have occurred by the instant t (i.e., $lst(tp_i) \leq t$);
- E_t: the set of events tp_i which can occur at time t (i.e., $est(tp_i) \leq t < lst(tp_i)$);
- A_t: the set of events tp_i which have to occur after time t (i.e., $est(tp_i) > t$).

where an event tp_i is either the start time or the end time of an activity a_l and, given a resource r_k, it has associated a resource usage value ru_{ik} respectively equal to $-req_{lk}$ and req_{lk}.

Based on this partition, the contribution of any time point tp_i to the maximum (minimum) resource value can be computed according to which of the three sets the event tp_i belongs to. In fact, since the time points in B_t are those which happen before or at time t, they all contribute - with the associated resource usage ru_{ik} - to the value of the resource profile of r_k in the instant t. By the same argument we can exclude from this computation the events in A_t, as they happen after t. Thus it is evident that the maximum (minimum) resource value depends on the maximum (minimum) contribution that the events in E_t may give: the maximum resource usage at time t is:

$$L_k^{max}(t) = \sum_{tp_i \in B_t} ru_{ik} + \sum_{tp_i \in P_{max}(E_t)} ru_{ik} \quad (4)$$

where $P_{max}(E_t)$ is a subset of E_t which gives the maximum contribution that any combination of elements in E_t can give. Thus a fundamental step is to calculate the subset $P_{max}(E_t)$: a trivial and, unfortunately, expensive approach consists of enumerating all possible combinations. Indeed this approach might require exponential CPU-time to compute the resource bounds which makes this approach unrealistic. A method to overcome this problem has been introduced in [25] where the author describes a polynomial algorithm to compute the set P_{max} (P_{min}).

Integration of the envelope computation into a PCP algorithm is quite natural. It can be used to restrict resource profile bounds, in accordance with the current temporal constraints in the underlying STP. The advantage of using the resource envelope is that all possible temporal allocations are taken into account at each step of the solving process. In the remainder of this section we describe how peak detection has been performed starting from an envelope.

6.1.1. Detecting peaks on resource envelopes

Envelope characteristics can be used as a means to analyze the current situation for possible flaws. Indeed, if the value of the resource envelope does not respect the capacity constraints then the current situation is not admissible, or more precisely, there exists at least a temporal solution which is not feasible. At this point it is necessary to extract from the current situation the set of activities or *contention peak* which leads to a resource conflict.

In [29] we introduced a collection method based on the analysis of sets B_t, E_t, A_t, and $P_{max}(E_t)$. It also assumes that each activity simply uses resources; without production and/or consumption. It is worth noting that the sets above are collected during the resource envelope computation thus their use does not imply any additional computational overhead.

In this method the two first steps to collect contention peaks consist of computing the resource envelope and matching it with the resource capacity to find

possible intervals of over-allocation. Then in the following step, the contention peaks are collected according to the following rules:

1. pick any activity a_i such that the time point associated with its start time is in $P_{max}(E_t)$ while the time point associated with its end time is not in $P_{max}(E_t)$, that is:
$$\{a_i | st_i \in P_{max}(E_t) \wedge et_i \notin P_{max}(E_t)\}.$$
2. to avoid collecting redundant contention peaks, the collection of a new contention peak is performed only if:
   (a) there exists at least an activity a_i such that the time point associated to its end time, et_i, moves from A_{t-1} to $B_t \cup E_t$ (i.e. $et_i \in A_{t-1} \cap (B_t \cup E_t)$);
   (b) there exists at least an activity a_j such that the time point associated to its start time, st_j, moved in to P_{max} since the last time a conflict peak has been collected.

The first rule is necessary to identify if an activity a_i is contributing to the value of $L_k^{max}(t)$. In fact when its end time will belong to P_{max} the start time will belong to $P_{max} \cup B_t$. Thus the effects of the two events will be balanced out, giving a void contribution to the resource envelope value.

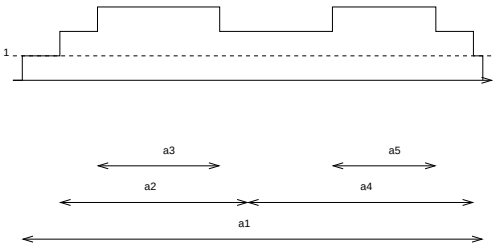


Fig. 6. Resource Envelope: detection of maximal peaks

The second rule is necessary to avoid the collection of redundant peaks. Let us consider the example in Fig. 6: here there are five activities each one requiring the same binary resource. The arrows represent the possible interval of allocation of each activity. If peaks were collected considering only the interval of over allocation, we would have the following result: $\{a_1, a_2\}$, $\{a_1, a_2, a_3\}$, $\{a_1, a_4\}$ and $\{a_1, a_4, a_5\}$. It is possible to note that the first and the third set are subsets of, respectively, the second and the fourth. Considering instead the changes into both A_t and B_t, we are able to compute non-redundant sets; in the case of the example $\{a_1, a_2, a_3\}$ and $\{a_1, a_4, a_5\}$.

6.2. Inside-Out Generation Using Early Start Profiles

A quite different analysis of resource profiles has been proposed in [6]. In that paper an algorithm called ESTA (for Earliest Start Time Algorithm) was first proposed which reasons with the earliest start time profile:

Definition 3 (Earliest Start Time Profile) *Let be $est(tp_i)$ the earliest time value for the time point tp_i. For each resource r_k we define the Earliest Start Time Profile as the function:*

$$Q_k^{est}(t) = \sum_{\forall tp_i : est(tp_i) \leqslant t} ru_{ik}$$

This method computes the resource profile according to one precise temporal solution: the Earliest Start Time Solution, that is, the solution obtained by allocating each activity to its temporal lower bound. This can be easily computed when an activity network is given by just computing the shortest path distance between the temporal source and each activity of the problem (for further details see [12]). This method exploits the fact that unlike the Resource Envelope, it analyzes a well-defined scenario instead of the range of all possible temporal behaviors.

It is worth noting that the key difference between the earliest start time approach with respect to the resource envelope approach is that while the latter gives a measure of the worst-case hypothesis, the former identifies "actual" conflicts in a particular situation (earliest start time solution). In other words, the envelope-based approach considers what *may* happen in such a situation relative to the entire set of possible solutions; the earliest start-time approach, instead, considers what *will* happen in such a particular case.

As in the case of the resource envelope analysis in the particular case of activities that only use resources, we can use a peak detection approach that avoids sampling peaks which are subsets of other peaks. Specifically, we follow a strategy of collecting sets of maximal peaks, i.e., sets of activities such that none of the sets is a subset of the others (see [8] for a detailed description).

The limitation of this approach with respect to our broader purpose of producing a $\mathcal{POS}$ is that it ensures resource-consistency of only one solution of the problem, the earliest start time solution. Using a PCP computation for solving, we always have a set of temporally consistent solutions S_T. However, ESTA will not synthesize a set of solutions for the problem (i.e., $S_T \nsubseteq \mathcal{S}$), but the single solution in the earliest start time of

Algorithm 2 Chaining procedure

Input: a problem P and one of its fixed-time schedules S
Output: A partial order solution $\mathcal{POS}$
  $\mathcal{POS} \leftarrow P$
  Sort all the activities according to their start times in S
  Initialize the all chains with the dummy activity a_0
  for all resource r_k **do**
    for all activity a_i **do**
      for 1 **to** req_{ik} **do**
        $chain \leftarrow$ SELECTCHAIN(a_i, r_k)
        $a_k \leftarrow last(chain)$
        $\mathcal{POS} \leftarrow \mathcal{POS} \cup \{a_k \prec a_i\}$
        $last(chain) \leftarrow a_i$
  return $\mathcal{POS}$

the resulting STP. Below, we describe a method for overcoming this limitation and generalizing an early start time solution into a partial ordered schedule. This will enable direct comparison with the $\mathcal{POS}$ produced by the envelope-based approach.

6.2.1. Producing a POS with Chaining

A first method for producing flexible solutions from an early start time solution has been introduced in [6]. It consists of a flexible solution where a *chain* of activities is associated with each unit of each resource.

In this section we generalize that method for the more general RCPSP/max scheduling problem considered in this paper (see Algorithm 2). Given a earliest start solution, a transformation method, named *chaining*, is defined that proceeds to create sets of chains of activities. This operation is accomplished by deleting all previously posted leveling constraints and using the resource profiles of the earliest start solution to post a new set of constraints. The basic idea is to "stretch" earliest start time solution into a set of solutions consistent with the initial problem constraints.

The first step is to consider a resource r_k with capacity cap_k as a set R_k of cap_k single capacity sub-resources. In this light the second step is to ensure that each activity is allocated to the same subset of R_k. This step can be viewed in Fig. 7: on the left there is the resource profile of a resource r_k, each activity is represented with a different color. The second step maintains the same subset of sub-resources for each activity over time. For instance, in the center of Fig. 7 the light gray activities are re-drawn in the way such that they are always allocated on the fourth sub-resource. The last step then is to build a chain for each sub resource in R_k. On the right of Fig. 7 this step is represented by the added constraints. This explains why the second step is needed. Indeed if the chain is built taking into account only the resource profile, there can be a problem with the relation between the light gray activity and the white one. In fact, using the chain building procedure just described, one should add a constraint between them, but that will not be sound. The second step allows this problem to be avoided, taking into account the different allocation on the set of sub-resources R_k.

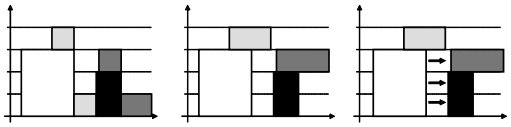


Fig. 7. Chaining method: intuition

Algorithm 2 uses a set of queues, or chains, to represent each capacity unit of the resource r_k. The algorithm starts by sorting the set of activities according to their start time in the solution S. Then it proceeds to allocate the capacity units needed for each activity. It selects only the capacity units available at the start time of the activity. Then when an activity is allocated to a chain, a new constraint between this activity and the previous one in the queue is posted. Let m and $max_{cap} = \max_{k=1}^{m} cap_k$ be respectively the number of resources and the maximum capacity among the resources, the complexity of the chaining algorithm is $O(n \log n + n \cdot m \cdot max_{cap})$.

6.3. Summary of PCP Algorithm Variants

In closing the section we remark again that by working with different resource profiles we have created two orthogonal approaches to generating a $\mathcal{POS}$: EBA (from Envelope Based Algorithm) and ESTA. One of them has required a post processing phase to be adapted to the current purpose (from the adaptation, the name ESTA^C). Given these two basic PCP configurations, recall that conflicts can be extracted from peaks according to three different strategies: pairwise selection, MCS linear sampling and MCS quadratic sampling. The combination of these three methods with the two different approaches to maintaining resource information thus leads to six different configurations: three based on the resource envelope, EBA, EBA+MCS linear, EBA+MCS quadratic, and three based on the earliest start time profile, ESTA^C, ESTA^C+MCS linear, ESTA^C+MCS quadratic. The next section presents a discussion of the results obtained testing the six approaches on a significant scheduling problem benchmark: RPCSP/max.

7. Experimental Analysis

In this section we provide an empirical evaluation of the methods introduced above. First of all we introduce and analyze metrics to capture desirable properties of robust partial order schedules. Once a pair of metrics is introduced we analyze the results obtained applying the methods on different RCPSP/max benchmarks.

7.1. Metrics to Compare Partial Order Schedules

It is intuitive that the quality of a partial order schedule is tightly related to the set of solutions that it represents. In fact the greater the number of solutions, the greater the expected ability of facing scheduling uncertainty. Furthermore, another aspect to consider in the analysis of the solutions clustered into a partial order schedule is the distribution of these alternatives. This distribution will be the result of the configuration given by the constraints present in the solution.

A first measure to consider the aspects above is taken from [1]. In this work the authors describe a metric, named $flex$, that can be defined as it follows:

$$flex = \frac{|\{(a_i, a_j)|a_i \not\prec a_j \wedge a_j \not\prec a_i\}|}{n(n-1)} \quad (5)$$

where n is the number of activities and the set $\{(a_i, a_j)|a_i \not\prec a_j \wedge a_j \not\prec a_i\}$ contains all the pairs of activities for which no precedence relation is defined. This measure counts the *number of pairs of activities in the solution which are not reciprocally related by simple precedence constraints*. It provides a first analysis of the configuration of the solution. The rationale is that when two activities are not related it is possible to move one without moving the other one. Hence, the higher the value of $flex$ the lower the degree of interaction among the activities.

Unfortunately $flex$ is able to give only a qualitative evaluation of the solution. In fact it considers only whether a "path" exists between two activities, not how long it is. Even though this may be sufficient for a scheduling problem with no time lag constraints like the one used in [1], in a problem like the RCPSP/max it is necessary to integrate the flexibility measure described above with an other one that also takes into account this quantitative aspect of the problem (or solution).

Before introducing a different measure, it is worth noting that in order to compare two or more $\mathcal{POS}$s it is also necessary to have a finite number of solutions. This is possible assuming that all the activities in a given problem must be completed within a specified, finite, horizon. *Hence, it follows that within the same horizon H, the greater the number of solutions represented in a $\mathcal{POS}$, the greater its robustness*. The goal then is to compute a fair bound (or horizon) that does not introduce any bias in the evaluation of a solution (therefore a time interval that allows all the activities to be executed). A possible alternative is the following:

$$H = \sum_{i=1}^{n} p_i + \sum_{\forall(i,j)} l_{ij}^{min} \quad (6)$$

that is, the sum of all activity processing times p_i plus the sum of all the minimal time lags l_{ij}^{min}. The minimal time lags are taken into account to guarantee the minimal distance between pairs of activities. In fact considering the activities in a complete sequence (i.e., the sum of all durations) may not be sufficient.

The presence of a fixed horizon allows introduction of a further metric taken from [6]: this is defined as the average width, relative to the temporal horizon, of the temporal slack associated with each pair of activities (a_i, a_j):

$$fldt = \sum_{i=1}^{n} \sum_{j=1 \wedge j \neq i}^{n} \frac{slack(a_i, a_j)}{H \times n \times (n-1)} \times 100 \quad (7)$$

where H is the horizon of the problem defined above, n is the number of activities, $slack(a_i, a_j)$ is the width of the allowed distance interval between the end time of activity a_i and the start time of activity a_j, and 100 is a scaling factor.[3] 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 involving an activity is absorbed by the temporal flexibility of the solution instead of generating deleterious domino effects (the higher the value of $fldt$, the lower the risk, i.e., the higher the probability of localized changes).

In order to produce an evaluation of the two criteria $fldt$ and $flex$ that is independent from the problem dimension, we present the normalization of the results according to an upper bound. The latter is obtained for each metric $\mu()$ considering the value $\mu(\mathcal{P})$ that is the quality of the initial network that represents the temporal aspect of the problem. In fact, for each metric the addition of precedence constraints between activities that are necessary to establish a resource-consistent solution can only reduce the initial value $\mu(\mathcal{P})$. Then the

[3] In the original work this was defined as robustness, using the symbol RB, of a solution.

normalized value for the solution S of the problem $\mathcal{P}$ will have the following form:

$$|\mu(S)| = \frac{\mu(S)}{\mu(\mathcal{P})} \quad (8)$$

where the higher $|flex|$ or $|fldt|$ value the better.

7.2. Results Analysis

In this section we present a comparison of each algorithm on the benchmark problems defined in [20]. This benchmark consists of four sets j10, j20, j30, and j100. The first three are composed of 270 problem instances of different size 10×5, 20×5 and 30×5 (number of activities $\times$ number of resources). The benchmark j100 is instead composed of 540 instances each of 100 activities and 5 resources.

The results obtained are subdivided according to the benchmark set in Table 1. First, we observe that none of the six tested strategies are able to solve all the problems in the benchmark sets (% column). This is possible because all the strategies are based on an incomplete search schema. To take into account these different solving capabilities the rest of the experimental results are computed with respect to the subset of problem instances solved by all the six approaches.

In the following results analysis we distinguish two complementary aspects of the different techniques: (1) the capabilities of the solving process and (2) the quality of the solutions obtained.

Solving capabilities. As shown before we have different capabilities in terms of number of solved problem with respect to theEBA and ESTA^C variants. In particular it is worth noting the lack of scalability of the EBAs. The reader can see that we have 97.78% (best EBA result) vs. 98.15% (best ESTA^C result) in the case of j10, 89.63% vs 96.67% in the case of j20, 43.33% vs. 97.04% in the case of j30, and 27.04% vs. 99.26% in the case of j100.

Another aspect to consider in evaluating solving process capabilities is the required CPU-time (shown in seconds). The results obtained for the EBA variants are worse than the CPU values obtained for ESTA^{C}s. In fact the larger the benchmark size the larger the difference in required CPU times. The results range from a ratio of 5.5 (in the case of j10) to one of about 400 (in the case of j100).

These facts induce further observations about the basic strategies behind the two algorithms. EBA removes all possible resource conflicts from a problem $\mathcal{P}$ by posting precedence constraints and relying on an envelope computation that produces the *tightest* possible resource-level bounds for a flexible schedule. When these bounds are less than or equal to the resource capacities, we have a resource-feasible partial order ready to *face* executional uncertainty. However, in order to remove all possible conflicts EBA has to impose more precedence constraints than does ESTA^C (see columns labeled with *pc*, posted constraints), with the risk of overcommitment in the final solution.

Solution qualities. For the evaluation of the solution qualities we have taken into account three different measures: $flex$, $fldt$, and mk.

The first two are directly correlated to solution robustness. Considering these measures in the case of the smaller benchmarks j10, j20, and j30, we have a very small difference in general between the results obtained with EBAs and ESTA^{C}s, even though the ESTA^C variants present a better behavior. A completely different result comes considering the benchmark j100: in this case the EBA variants seem more suitable than the ESTA^{C}s, for obtaining robust solutions. Notwithstanding in evaluating this result, we have to keep in consideration also the different solving capabilities (27.04% vs 99.26%) and the fact that this evalution is done on the subset of common solved instances. Therefore it seems to us that the number of instances taken into account are too few to make strong claims in the case of the benchmark j100.

On the other hand, as previously explained, ESTA^C is a two step procedure: the ESTA step creates a particular partial order that guarantees only the existence of the early start time solution; the chaining step converts this partial order into a $\mathcal{POS}$. It is worth reminding that the number of precedence constraints is always $O(n)$ and for each resource, the *form* of the partial order graph is a set of *parallel* chains. These last observations probably identify the main factors which enable a more robust solution behavior, i.e., ESTA^C solutions can be seen as a set of *layers*, one for each unit of resource capacity, which can *slide* independently to hedge against unexpected temporal shifts.

The last aspect we present in this evaluation is the makespan of the solutions. First of all it is necessary to clarify what we mean for makespan of a partial order schedule. In fact a $\mathcal{POS}$ represents several fixed-time schedules each with a different makespan. In this case we consider the minimum makespan among the makespans of the solutions in the $\mathcal{POS}$, i.e. the makespan of the earliest start time solution of the $\mathcal{POS}$. The results in Table 1 show as the ESTA^C variants outperform the EBA ones. In particular we can

Table 1
Results on the four benchmarks

	j10						j20					
	%	**mk**	**cpu**	**pc**	**|flex|**	**|fldt|**	**%**	**mk**	**cpu**	**pc**	**|flex|**	**|fldt|**
EBA	77.04	58.31	0.11	11.54	0.14	0.63	50.74	96.48	1.37	33.40	0.16	0.64
EBA+MCS linear	85.19	55.29	0.18	11.12	0.17	0.65	71.11	92.65	1.83	32.87	0.15	0.60
EBA+MCS quadratic	97.78	55.47	0.19	12.38	0.16	0.65	89.63	94.03	1.99	34.98	0.13	0.58
ESTA^C	96.30	47.35	0.02	6.40	0.19	0.67	95.56	72.90	0.12	18.69	0.20	0.65
ESTA^C+MCS linear	98.15	46.63	0.03	6.23	0.20	0.66	96.67	72.45	0.18	17.49	0.20	0.65
ESTA^C+MCS quadratic	98.15	46.70	0.03	6.26	0.20	0.68	96.67	72.75	0.19	17.40	0.19	0.64
	j30						**j100**					
	%	**mk**	**cpu**	**pc**	**|flex|**	**|fldt|**	**%**	**mk**	**cpu**	**pc**	**|flex|**	**|fldt|**
EBA	43.33	118.17	7.53	63.29	0.23	0.69	21.11	501.61	33.00	53.67	0.13	0.72
EBA+MCS linear	68.89	112.14	8.82	56.84	0.18	0.59	26.11	606.31	78.64	73.41	0.13	0.58
EBA+MCS quadratic	82.22	116.10	10.94	59.64	0.16	0.56	27.04	632.27	183.79	76.88	0.11	0.56
ESTA^C	96.30	79.21	0.41	35.10	0.25	0.60	99.26	374.24	0.48	68.47	0.07	0.50
ESTA^C+MCS linear	96.67	78.45	0.74	34.07	0.26	0.62	99.44	374.22	0.48	67.94	0.07	0.50
ESTA^C+MCS quadratic	97.04	78.55	0.83	34.00	0.25	0.64	99.26	374.35	0.48	68.18	0.07	0.50

note that the larger the problem size the larger the gap between the two methodologies, e.g., in the case of benchmark j100 we have an average value of 501.61 for the best EBA variant against 374.22 obtained with ESTA^C + MCS linear.

8. Discussion: the Solve & Robustify model

In this paper, different needs – and, dually, different trade-offs between quality and computational times – are addressed by different algorithms or combinations of solving techniques in a meta-heuristic schema. The results in Table 1 have shown the ESTA^C variants be better suited to generating robust solutions. Even though EBA methods directly produce partial order schedules considering this aim at each step of the solving process, the ESTA^C methods yield solutions that are no less robust (in terms of the measures $flex$ and $fldt$). Moreover these methods show better performance when problem size is increased. They are able, in fact, to limit the increase in CPU times and maintain a high percentage of solved instances.

This two-step approach, which we refer to as *Solve & Robustify*, is based on the assumption of independence between the classical scheduling objectives – like minimizing the makespan – and the need to obtain a robust (or flexible) solution. Even though this assumption is in general not true, the two step approach can allow exploitation of *state-of-the-art* schedulers to obtain optimal solutions with respect to the classical objectives. Then, in a next step, a flexible solution can be generated trying to preserve the optimality of the starting schedule. Preserving optimality can be very important when a low degree of uncertainty is present. In this case the actual execution of the problem remains "close" to the *expected value problem* (i.e., the problem as described in input). Therefore, the characteristics of a $\mathcal{POS}$ tend to maintain, due to necessary repairs, a new allocation of the activities that is close to the original schedule. It is intuitive that the closer the two allocations are the less loss that is incurred in objective function values.

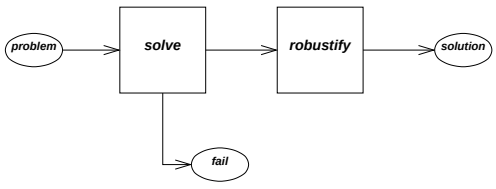


Fig. 8. Solve & Robustify model

Figure 8 shows a possible sketch of the two step approach. This model is based on the use of two separate modules: a greedy solver that has the aim of finding an initial fixed-time solution, and a "robustify" module where a partial order schedule is synthesized from this solution. Different variants of the two step approach can be obtained by different combinations of the two modules. As the figure highlights, only in the first phase, in which the search for a solution occurs, is it possible to fail (i.e., when it is not possible to find a

first fixed-time schedule). In the Robustify step in contrast, whenever a starting schedule exists, it is always possible to generate a $\mathcal{POS}$.

A further remark concerns the generation of flexible schedules. In the case of multi-capacitive resource problems, we have in general more than one possible $\mathcal{POS}$ corresponding to a fixed-time schedule. On the contrary in the case of binary resources, i.e. job shop problem, a fixed-time solution also gives a unique "linearization" of all the activities. This aspect of multi-capacitive resource problems has supported and suggested the idea of exploring the space of possible $\mathcal{POS}$s obtainable from the same fixed-time schedule with the aim of increasing robustness characteristics [28,30]. A different chaining algorithm is described in [19] where the authors use $\mathcal{POS}$s and their flexibility characteristics to apply a local search approach with the aim of optimizing schedule makespan.

9. Conclusion

Research in constraint-based scheduling has typically formulated the problem as one of finding a consistent assignment of start times for each goal activity. In contrast, we are investigating approaches to scheduling that operate with a problem formulation more akin to least-commitment frameworks: the Precedence Constraint Posting procedure. In this approach the goal is to post sufficient additional precedence constraints between pairs of activities contending for the same resources to ensure feasibility with respect to time and capacity constraints. Solutions generated in this way generally represent a set of feasible schedules by using a temporally flexible graph.

Exploiting this flexibility, in this work we have investigated two orthogonal PCP approaches (EBA and ESTA^C) to building scheduling solutions that hedge against unexpected events. The two approaches are based on two different methods for maintaining profile information: one that considers all temporal solutions (the resource envelope) and one that analyzes the profile for a precise temporal solution (the earliest start time solution).

To evaluate the quality of respective solutions we introduced two measures that capture desirable properties of robust solutions, $fldt$ and $flex$, correlated to the degree of schedule robustness that is retained in generated solutions. Considering comparative performance on a set of benchmark project scheduling problems, we have shown that the two step ESTA^C procedure, which first computes a single-point solution and then translates it into a temporally flexible partial order schedule, is a more effective approach than the pure, least-commitment EBA approach. In fact, the first step preserves the effectiveness of the ESTA approach, while the second step has been shown to be capable of reinstating temporal flexibility in a way that produces a final schedule with better robustness properties.

Acknowledgements

Stephen F. Smiths work is supported in part by the National Science Foundation under contract #9900298, by the Department of Defense Advanced Research Projects Agency under contract #FA8750-05-C-0033 and by the CMU Robotics Institute. Amedeo Cesta, Angelo Oddi and Nicola Policella's work is partially supported by MIUR (Italian Ministry for Education, University and Research) under project ROBOCARE (L. 449/97) and contract #2005-015491 (PRIN).

References

[1] M. A. Aloulou and M. C. Portmann. An Efficient Proactive-Reactive Scheduling Approach to Hedge Against Shop Floor Disturbances. In G. Kendall, E. Burke, S. Petrovic, and M. Gendreau, editors, *Multidisciplinary Scheduling: Theory and Applications*, pages 223–246. Springer, 2005.

[2] P. Baptiste and C. Le Pape. A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling. In *Proceedings of the* 14^{th} *International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 600–606. Morgan Kaufmann, 1995.

[3] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*, volume 39 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, 2001.

[4] 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.

[5] J. C. Beck, E. D. Davenport, A. J. Davis, and M. S. Fox. The ODO Project: Towards a Unified Basis for Constraint-Directed Scheduling. *Journal of Scheduling*, 1:89–125, 1998.

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

[7] A. Cesta, A. Oddi, and S. F. Smith. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the* 16^{th} *International Joint Conference on Artificial Intelligence, IJCAI-99*, pages 1022–1029. Morgan Kaufmann, 1999.

[8] A. Cesta, A. Oddi, and S. F. Smith. A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristics*, 8(1):109–136, January 2002.

[9] A. Cesta and C. Stella. A Time and Resource Problem for Planning Architectures. In S. Steel and R. Alami, editors, *Proceedings of the 4^{th} European Conference on Planning, ECP-97*, volume 1348 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 1997.

[10] C. Cheng and S. F. Smith. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings of the 12^{th} National Conference on Artificial Intelligence, AAAI-94*, pages 1086–1091. AAAI Press, 1994.

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

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

[13] R. Dechter and F. Rossi. Constraint Satisfaction. In L. Nadel, editor, *Encyclopedia of Cognitive Science*. Nature Publishing Group, 2002.

[14] B. Drabble and A. Tate. The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner. In *Proceedings of the 2^{nd} International Conference on Artificial Intelligence Planning Systems, AIPS-94*, pages 243–248. AAAI Press, 1994.

[15] M. Drummond, J. Bresina, and K. Swanson. Just-in-Case Scheduling. In *Proceedings of the 12^{th} National Conference on Artificial Intelligence, AAAI-94*, pages 1098–1104. AAAI Press, 1994.

[16] H. H. El Sakkout and M. G. Wallace. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4):359–388, 2000.

[17] M. S. Fox. Constraint Guided Scheduling: A Short History of Scheduling Research at CMU. *Computers and Industry*, 14(1–3):79–88, 1990.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[19] D. Godard, P. Laborie, and W. Nuitjen. Randomized Large Neighborhood Search for Cumulative Scheduling. In *Proceedings of the 15^{th} International Conference on Automated Planning & Scheduling, ICAPS'05*, pages 81–89. AAAI Press, 2005.

[20] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark Instances for Project Scheduling Problems. In J. Weglarz, editor, *Project Scheduling - Recent Models, Algorithms and Applications*, pages 197–212. Kluwer Academic Publishers, Boston, 1998.

[21] V. Kumar. Algorithms for Constraint-Satisfaction Problems:A Survey. *Artificial Intelligence Magazine*, 13(1):32–44, 1992.

[22] P. Laborie. Algorithms for Propagating Resource Constraints in A.I. Planning and Scheduling: Existing Approaches and New Results. *Artificial Intelligence*, 143(2):151–188, 2003.

[23] P. Laborie and M. Ghallab. Planning with Sharable Resource Constraints. In *Proceedings of the 14^{th} International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 1643–1651. Morgan Kaufmann, 1995.

[24] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[25] N. Muscettola. Computing the Envelope for Stepwise-Constant Resource Allocations. In *Principles and Practice of Constraint Programming, 8^{th} International Conference, CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 139–154. Springer, 2002.

[26] W. Nuijten and C. Le Pape. Constraint-Based Job Shop Scheduling with Ilog-Scheduler. *Journal of Heuristics*, 3(4):271–286, March 1998.

[27] W. P. M. Nuijten and E. H. L. Aarts. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

[28] N. Policella, A. Oddi, S. F. Smith, and A. Cesta. Generating Robust Partial Order Schedules. In M. Wallace, editor, *Principles and Practice of Constraint Programming, 10^{th} International Conference, CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 496–511. Springer, 2004.

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

[30] Nicola Policella. *Scheduling with Uncertainty: A Proactive Approach using Partial Order Schedules*. PhD thesis, Department of Computer and Systems Science, University of Rome "La Sapienza", March 2005.

[31] B. Roy and B. Sussman. Les problemes d'ordonnancement avec contraintes disjonctives. Note DS n. 9 bis, SEMA, Paris, 1964.

[32] N. M. Sadeh. *Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh PA, March 1991.

[33] S. F. Smith. OPIS: A Methodology and Architecture for Reactive Scheduling. In M. Fox and M. Zweben, editors, *Intelligent Scheduling*, pages 29–66. Morgan Kaufmann, 1994.

[34] S. F. Smith and C. Cheng. Slack-based Heuristics for Constraint Satisfactions Scheduling. In *Proceedings of the 11^{th} National Conference on Artificial Intelligence, AAAI-93*, pages 139–144. AAAI Press, 1993.

[35] E. P. K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.