# Robust Scheduling: Analysis and Synthesis of flexible solutions

**Technical Report** · October 2003

CITATIONS
3

READS
35

**1 author:**

Nicola Policella
European Space Agency
**115** PUBLICATIONS **749** CITATIONS

SEE PROFILE

# Robust Scheduling: Analysis and Synthesis of flexible solutions

Thesis Proposal

*Nicola Policella*

Dipartimento di Informatica e Sistemistica
Universita' di Roma "La Sapienza"
policella@dis.uniroma1.it

**Abstract**

In most practical scheduling environments, off-line schedules can have a very limited lifetime and scheduling is really an ongoing 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.

We are pursuing the idea of promoting robust response through the generation of flexible schedules – schedules that encapsulate a set of possible execution futures and hence can accommodate some amount of executional uncertainty. At this stage, our particular focus is generation of schedules that retain temporal flexibility.

## Contents

# 1 Introduction

In the realm of scheduling problems different sources of uncertainty can arise: durations may not be known, resources may have lower capacity than expected (i.e., machine breakdown), new tasks may need to be taken into account. Given this, one highly desirable characteristic of a schedule is that the reactions to unexpected events during execution entail small and localized changes.

One way to face this problem consists of using on-line (reactive) approaches. These approaches try to repair the schedule each time a new disruption happens. Keeping the pace with execution requires that the repair process be both fast and complete. A repair must be fast because of the need to re-start execution of the schedule as soon as possible. A repair also has to be complete in the sense that it has to take into account all changes that have occurred, avoiding to produce new ones. As these two goals can be conflicting a compromise solution is often needed. Different approaches exist and they tend to favor either the swiftness of their reaction [Smith, 1994] or the completeness of the new solution [El Sakkout and Wallace, 2000].

Alternative approaches to managing execution in dynamic environments have focused on building schedules that retain flexibility and hedge against uncertainty (off-line or proactive approaches). Robust approaches aim at building solutions able to absorb some level of unexpected event without rescheduling. To achieve such a feature, different techniques have been investigated. One consists of building redundancy-based solutions, both of resources and of time, taking into account the uncertainty present in the domain [Davenport et al., 2001]. An alternative technique is to construct a set of contingencies (i.e., a set of different solutions) and use the most suitable with respect to the actual evolution of the environment [Drummond et al., 1994]. An important point to note is that both types of approaches above need to be aware of the possible events that can occur in the environment. In some cases, this need for knowledge about the uncertainty in the operating environment can present a barrier to their use.

For this reason, in the perspective of robust approaches, we consider a less knowledge intensive approach: to simply build solutions that retain temporal flexibility where problem constraints allow. In [Ginsberg et al., 1998] a similar concept of producing solutions that promote bounded, localized recovery from execution failures is also proposed. The two conditions above are desired to insure an ability to keep pace with execution and, at the same time, maintain stability in the solution. To achieve these features, the idea is to construct partially ordered solutions, by introducing ordering constraints to resolve resource conflicts between pairs of activities. By providing greater execution flexibility, such solutions are more advantageous than fixed-time schedules (where precise start and end times are assigned to all activities). Fixed-time schedules are quite brittle and it is typically very difficult to follow them exactly during execution. Moreover, a flexible solution allows explicit reasoning about the uncontrollability of external events and the ability to include execution countermeasures.

# 2 Background

In this section, we describe important concepts and techniques that will be used frequently throughout this proposal. These are all based on the Constraint Satisfaction Problem paradigm.

## 2.1 Constraint Satisfaction Problem

We consider the Constraint Satisfaction Problem formalism, CSP, since it has the generality we desire to model the scheduling problem. A CSP is a 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, that is, $v_1 \in D_1$, $v_2 \in D_2$ and $v_n \in D_n$, and
- $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, a set of $m$ constraints, $c_k(v_1, v_2, \ldots, v_n)$, which are predicates defined on the Cartesian product $D_1 \times D_2 \times \ldots \times D_n$.

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

$$\{\lambda_1, \lambda_2, \ldots, \lambda_n\} \subseteq D_1 \times D_2 \times \ldots \times D_n$$

such that the whole set of constraints is satisfied.

An instance of a CSP $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ 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.

## 2.2 Scheduling Problem

A scheduling problem consists in, given a set of activities that require a set of resources, finding an allocation of each activity such that all the resources are used consistently.

A common representation by the CSP paradigm, uses to extract from the overall problem the sub problem concerning the time constraints (temporal constraints network). Such a formalism allows to exploit the polynomially aspect of such a sub problem. Moreover, let $\mathcal{S}$ the set of solution of the overall problem and $S_T$ the set of solution of the temporal constraints network, the following relation $\mathcal{S} \subseteq S_T$ is held.

The temporal constraints network is represented as a Simple Temporal Problem (STP). For each activity $a_i$ two relevant events are identified: the start time, $s_{a_i}$, and the end time, $e_{a_i}$. All these events create a set $\mathcal{T}$ of temporal variables $t_i$ named time points: at each activity $a_i$ are assigned the time points $t_{2i-1}$ and $t_{2i}$, for representing the two events, start and end time, of each activity. Moreover two dummy time points $t_0$ and $t_p$ (with $p = 2n + 1$) are used for representing the origin and the horizon of the problem (that is $t_0 \leqslant t_j \leqslant t_p \quad \forall j \in 1, \ldots, p-1$). The time constraints between time points represent both the duration of the activity, $dur_i \leqslant t_{2i} - t_{2i-1} \leqslant Dur_i$, and the constraints between pair of activities $d_{ij}^{min} \leqslant t_i - t_j \leqslant d_{ij}^{max}$. Figure 1 provides a synthesis of the representation for the case of two activities, $a_1$ and $a_2$, ordered by the constraints $d_{min} \leqslant a_2 - a_1 \leqslant d_{max}$. The activity $a_1$ uses two untis of the resource $r$ while the activity $a_2$ produces one unit of the same resource.

As it possible to see in Fig. 1, an STP can be associated with a directed edge-weighted graph $G_d = \{V_d, E_d\}$ named *distance graph* where the set of nodes $V_d$ represents the set of variables, or time points, $\{t_0, t_1, \ldots, t_p\}$ and the set of edges $E_d$ represents the set of constraints. According to the model above is possible to prove the following theorems:

Theorem 2.1: An STP is consistent if and only if its distance graph $G_d$ has no negative cycles.

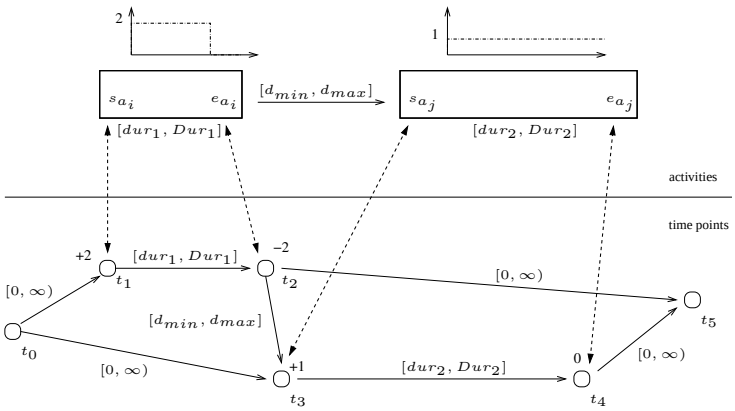


Fig. 1: Relation activity-time point

Corollary 2.1: Let $G_d$ be the distance graph of a consistent STP. Two consistent scenarios are given by

$$S_1 = \{d_{01}, d_{02}, \ldots, d_{0p}\}$$
$$S_2 = \{-d_{10}, -d_{20}, \ldots, -d_{p0}\}$$

which assign to each variable its latest and its earliest start time, respectively.

Given a set of resource $\mathcal{R}$ we define for each resource $r_j \in \mathcal{R}$ a level of availability $Q_j(t)$ that represents the available resource $r_j$ over time. For each resource $r_j \in \mathcal{R}$ an interval $[min_j, max_j]$ that bounds the value of $Q_j(t)$ is defined. At each time point $t_i$ a value $ru_{ij}$ that modifies the resource availability of $r_j$ is defined. Three are the possible behaviors: production ($ru_{ij} > 0$), consumption ($ru_{ij} < 0$) and no change of the resource availability ($ru_{ij} = 0$).

For a consistent assignment of the time points, $\{t_0 = \tau_0, t_1 = \tau_1, \ldots, t_p = \tau_p\}$, we define the resource profile as

Definition 2.1 (Resource Profile): Let $\mathcal{T}$ the set of time points and $\tau \in S_T$ a consistent assignment, for each resource $r_j$ we define the resource profile the function

$$Q_j^{\tau}(t) = \sum_{t_i \in \mathcal{T} \wedge \tau_i \leqslant t} ru_{ij}$$

Definition 2.1 allows to express the resource constraint as an n-ary constraint on the set of time points $\mathcal{T}$

Definition 2.2 (Resource Consistent): A STP solution $\tau \in S_T$ is resource consistent if and only if for each resource $r_j$ the following property holds:

$$min_j \leqslant \sum_{t_i \in \mathcal{T} \wedge \tau_i \leqslant t} ru_{ij} \leqslant max_j$$

## 2.3 Precedence Constraint Posting Approach

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. The constraint graph is an evolving representation of the search state, where a solution is a state with a single value remaining in the domain of each variable, and all constraints are satisfied (Algorithm 1).

**Algorithm 1** Constraint Posting Scheduling Procedures

**Input:** A problem $\mathcal{P}$
**Output:** A problem $\mathcal{S}$
$S_0 \leftarrow \mathcal{P}$
**loop**
  **if** termination condition are met **then**
    $\mathcal{S} \leftarrow S_0$
    EXIT
  **end if**
  **if** $S_0$ is not consistent **then**
    retract commitment(s)
    **if** no commitment to retract **then**
      FAILURE
    **end if**
  **else**
    make heuristic commitment $O(i, j, r)$ on $S_0$
  **end if**
**end loop**

PCP utilizes look-ahead analysis to dynamically direct (and redirect) an incremental search process. However, the ”variables” in PCP’s problem formulation are ordering decisions O(i,j,r) (for activities i and j contending for resource r) with two possible ”values”: i before j or j before i. From the standpoint of solution development, a constraint posting formulation of the problem can provide a more convenient search space in which to operate. During schedule generation, alternatives are not unnecessarily pruned by the need to (over) commit to specific start times. When the need for schedule revision becomes apparent, modifications can often be made much more directly and efficiently through simple adjustment of posted constraints.

## 3 Schedule Robustness

In the realm of scheduling problems different sources of uncertainty can arise: durations may not be known, resources may have lower capacity than expected (i.e., machine breakdown), new tasks may need to be taken into account. Given this, one highly desirable characteristic of a schedule is that the reactions to unexpected events during execution entail small and localized changes. In this section we tackle such an aspect providing a general definition of robust schedules and a more precise solution for a well defined domain. Last we introduce a set of metrics for evaluating the quality of such solutions.

### 3.1 Definitions

As we already told, one kind of approach consists in repairing the schedule using techniques ad hoc. We are pursuing a different approach concerns the generation of solutions able of facing possible disruptions. Such robust solutions can be defined as:

Definition 3.1 (Robust Schedule): A solution for a scheduling problem is robust if it has the ability of reacting to external events maintaining the solution as more stable as possible.

Two are the conditions used for defining the concept of robust schedule:

Reactivity An important aspect in the execution of a schedule is to respond immediately to unpredict evolution of the environment. Responding slowly to an external change can in practice lead to a failure of the execution.

Stability The solution has to avoid to amplify the effects of a change over all its components. Keeping a solution as more stable as possible insures notable advantages. For instance a schedule might involve many people, each with different tasks to do. Changing everyone's task may lead to much confusion.

The current work analyzes the possibility of facing temporal disruptions, for instance, an activity lasts more than expected and/or the execution of an activity is shifted forward in time. To achieve the features described above, the idea is to construct partially order solutions, by introducing ordering constraints to resolve resource conflicts between pairs of activities. Moreover to have a right behavior for hedging against the executional uncertainty, the set of temporal solutions, represented by the partial order solution, have to be at same time also solutions of the overall problem. We name such a kind of solution *Partial Order Schedule*:

Definition 3.2 (Partial Order Schedule): A Partial Order Schedule $\mathcal{POS}$ is a partial order among the set of the activities such that all possible temporal solution is also a consistent assignment, that is $\mathcal{POS} \subseteq \mathcal{S}$.

An important aspect of a $\mathcal{POS}$ is the ability of answering rapidly to external changes. Indeed to propagate the effects of a disruption over the solution is akin to propagate a change over a Simple Temporal Problem (polynomial time). On the other side build a new solution from scratch would have a high complexity, being a CSP NP-complete.

By providing greater execution flexibility, such solutions are more advantageous than fixed-time schedules (where precise start and end times are assigned to all activities). Fixed-time schedules are quite brittle and it is typically very difficult to follow them exactly during execution. Moreover, a flexible solution allows explicit reasoning about the uncontrollability of external events and the ability to include execution countermeasures. There are several advantages to this approach over so-called "fixed-times" scheduling. From the standpoint of solution use, generation of sets of feasible schedules provides a measure of robustness against executional uncertainty, allowing actual activity start time decisions to be delayed and minimizing the need for solution revision. In the analysis of a Partial Order Schedule two different features can be emphasized:

Topology Analyzing the topology of a solution is possible to extract the relations among the activity so its robustness. For instance whether in a solution no relation exists between each pair of activity then the solution is extremely robust. Indeed thanks to the complete disconnection among the parts of the solution any disruption will have local effect.

Size For size we mean the number of schedules that a solution holds. Indeed a Partial Ordered Schedule can be viewed as a box of schedules. In this case the greater the number of solution the higher the robustness of the solution, that is, the higher the ability to pick a tailored schedule for the actual evolution of the world.

In the next section we introduce a set of metrics for evaluating the quality of a $\mathcal{POS}$ in terms of the features described above.

## 3.2 Evaluation Criteria

A fundamental point related to the concepts introduced above is the need for metrics that characterize the quality of a solution, and in general, the extent to which a solution is suitable for the execution phase.

In [Aloulou and Portmann, 2003] the authors propose a measure $flex_{seq}$ equals to the number of non oriented edges in the transitive graph representing the partial order:

$$flex_{seq} = \sum_{i=1}^{N} \sum_{j>i \wedge d_{ij} \times d_{ji} < 0}^{N} \frac{1}{N \times (N-1)} \tag{1}$$

in which $N$ is the number of activities and $d_{ij}$ is the distance between the two activities. The value $d_{ij} \times d_{ji} < 0$ identifies that there is not any relation between the two activities $a_i$ and $a_j$. This metric provides an analysis of the configuration of the solution. Indeed the transitive graph underlines the pair of activities which are not ordered in the solution. So the higher the value the lower the degree of interaction among the activities.

In [Cesta et al., 1998] a metric[1] based on the temporal slack associated with each activity is introduced:

$$fldt = \sum_{h \neq l} \frac{|d(e_{a_h}, s_{a_l}) - d(s_{a_l}, e_{a_h})|}{H \times N \times (N-1)} \times 100 \tag{2}$$

in which $H$ is the horizon of the problem, $N$ is the number of activities and $d(tp_1, tp_2)$ is the distance between the two time points. This metric aims at measuring the ***fluidity*** of a solution, i.e., the ability to use flexibility to absorb temporal variation in the execution of activities. The higher the value of $flex$, the less the risk of a "domino effect", i.e. the higher the probability of localized changes.

While the previous parameter measures the ability to avoid domino effects, another aspect of solution flexibility is the expected magnitude of potential changes. We introduce a further parameter for taking into account also the impact of disruptions on the schedule, or the ***disruptibility*** of a solution:

$$dsrp = \frac{1}{N} \sum_{i=1}^{N} Pr_{disr}(a_i) \times \frac{let(a_i) - eet(a_i)}{num_{changes}(a_i, \Delta_{a_i})} \tag{3}$$

where $Pr_{disr}(a_i)$ is the probability that a disruption occurs during the execution of the activity $a_i$. The value $let(a_i) - eet(a_i)$ represents the temporal flexibility of each activity $a_i$, i.e., the ability to absorb a change in the execution phase. The probability is considered because the flexibility of each activity gives a different contribution to the solution quality according to the possibility that a disruption can occur, or not, during its execution. Through the function $num_{changes}(a_i, \Delta_{a_i})$ the number of entailed changes given a right shift $\Delta_{a_i}$ of the activity $a_i$ is computed. The function equates to propagate forward the the value $\Delta_{a_i}$, counting the number of activities which are shifted (changes). In Sect. 4.5.1 both the probability distribution, $Pr_{disr}(a_i)$, and the right shift, $\Delta_{a_i}$, used in the empirical evaluation are described.

The intuition behind the disruptibility parameter consists of considering the trade-off between the flexibility of each activity, $let(a_i) - eet(a_i)$, and the number of changes implied, $num_{changes}(a_i, \Delta_{a_i})$. The latter can be seen as the price to pay for the flexibility of each activity.

[1] Named $rb$, robustness, in [Cesta et al., 1998].

**Algorithm 2** greedyPCP($\mathcal{P}$)

**Input:** a problem $\mathcal{P}$
**Output:** a solution $\mathcal{S}$

```
S_0 ← P
if Exists an unresolvable conflict in S_0 then
  FAILURE
else
  C_s ← Select-Conflict-Set(S_0)
  if C_s = ∅ then
    S ← S_0
  else
    (a_i{before}a_j) ← Select-Leveling-Constraint(C_s)
    S_0 ← S_0 ∪ {a_i{before}a_j}
    S ←PCP-greedy(S_0)
  end if
end if
```

## 4 Current work

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 planning frameworks, where 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 (i.e., the sets of activity start times that remain consistent with posted sequencing constraints), as opposed to a single assignment of start times.

The procedures we implemented is based on a constraint data-based. This core represents both the time and capacity constraints. The time constraints are represented as a Simple Temporal Problem (see 2.2). Based on the temporal constraints network further ordering between pair of activities are synthesized. They have the goal of pruning all the inconsistent allocation of activities on the resources. For analyzing the resource behavior is used the Resource Profile (Definition 2.1). As this is defined according to a single temporal solution there can be different approach with respect to the temporal solutions considered.

We define a PCP framework, Algorithm 2, that implements a simple greedy algorithm. Indeed at this stage our goal is to study the characteristics of possible approaches. Within this framework, in a PCP manner, a solution is produced 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.

Algorithm 2 shows the conflict removal procedure. Given a problem, in terms of a partial ordered plan, the first step consists of building an estimate of the resource levels needed. The analysis can highlight an infeasible current situation, where resource needs are greater then the availability. The function Select-Conflict-Set($S_0$) selects from a set of *contention peak* a set of pair $\langle a_i, a_j \rangle$ of activities or conflict, $C_s$, that identifies the possible conflicting situations. For solving such a case, a new precedence constraint, $a_i\{before\}a_j$, is synthesized and added to the problem.

The next sections will introduce two orthogonal approaches for representing and maintaining resource profile information, the resource envelope approach and the earliest start time approach. In the Sect. 4.4 the heuristics used for guiding the search will be introduced.

## 4.1 Comparing two opposite approaches

In the perspective of a "pure" least commitment a first approach takes into account all the possible temporal solutions $\tau \in S_T$. This is possible using the Resource Envelope as defined in [Muscettola, 2002]. According to the terminology introduced before we can define the Resource Envelope as

Definition 4.1 (Resource Envelope): Let $S_T$ the set of temporal solutions $\tau$, for each resource $r_j$ we define the Resource Envelope the two functions:

$$L_j^{max}(t) = \max_{\tau \in S_T} \{Q_j^\tau(t)\}$$

$$L_j^{min}(t) = \min_{\tau \in S_T} \{Q_j^\tau(t)\}$$

The use of the Resource Envelope allows to consider every temporal solutions exclusively.

For evaluating the ability of this approach to find flexible solutions we compare it with an orthogonal method to estimate the resource needs: the earliest start time profile [Cesta et al., 1998].

Definition 4.2 (Earliest Start Time Profile): Let $est_{t_i}$ the earliest start time for the time point $t_i$, for each resource $r_j$ we define the Earliest Start Time Profile the function:

$$Q_j^{est}(t) = \sum_{t_i \in \mathcal{T} \wedge est_{t_i} \leqslant t} ru_{ij}$$

This method computes the resource profile according to one precise temporal solution: the Early Start Time Solution (Corollary 2.1). The method exploits the fact that unlike the Resource Envelope, it analyzes a well-defined scenario instead of the range of all possible temporal behaviors. On the other side it will not sufficient, in general, to achieve a $\mathcal{POS}$ and for such a reason it will be necessary a post-processing step.

Before to introduce the details of the two approaches, we want to give an abstract picture of the two methods based on two different approaches for maintaining profile information, Fig. 2. On one side the resource envelope method equates to prune the space of temporal solution until it coincides to a set of solution of the overall problem. At the contrary using the earliest start time algorithm a first partial order solution is found. This may involve a set of solutions of the problem but it does not insure that all the temporal solutions involved are also consistent. For this reason an operator, chaining, will be applied for obtaining a partial order schedule.

## 4.2 The Resource Envelope Approach

The first method considered for guiding the search for reaching flexible solutions is the resource envelope defined in [Muscettola, 2002] and [Kumar, 2003]. These works prove it is possible to find the *tightest possible resource-level bound for a flexible plan* through a polynomial algorithm. The advantage of using the resource envelope is that

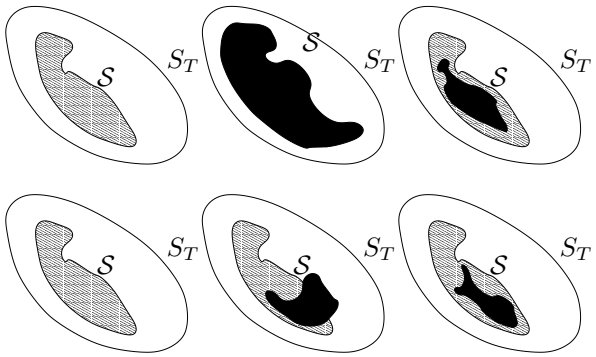


Fig. 2: High-level intuition behind the two kinds of approaches.

all possible temporal allocations are taken into account during the solving process. Thus, unlike the fixed time approaches, a solution consists of a set of feasible solutions. In the remainder of this section we briefly review the idea behind the computation of the resource envelope then we will describe the way in which it has been used to detect conflict peaks. We will conclude this section introducing a set of properties which allow to simplify the computation of the resource envelope.

To find the maximum (minimum) value of the resource level in an instant $t$ most methods subdivide the set of time points $\mathcal{T}$ (events) into the following subsets:

- $B_t$: the set of events $tp_i$ s.t. $let(tp_i) \leq t$;
- $E_t$: the set of events $tp_i$ s.t. $est(tp_i) \leq t < let(tp_i)$;
- $A_t$: the set of events $tp_i$ s.t. $est(tp_i) > t$.

Since the events in $B_t$ are those which will end before or at time $t$, they all contribute, with the associated resource usage $ru_{ik}(tp_i)$, to the value of the resource profile of $r_k$ in the instant $t$. By the same argument we can exclude from such a computation the events in $A_t$. Then the crucial point is to determine which of those in $E_t$ have to be take into account. A basic method consists of enumerating all the possible combinations of events in $E_t$. This method implies a high computational cost, and for this reason approximate techniques have been developed. In [Muscettola, 2002], instead, the author proves that to find the subset of $E_t$ for computing the upper (lower) bound, it is possible to avoid such an enumeration. The author shows that a polynomial algorithm can be found, taking into account the relations among the events through a reduction to a well-known tractable problem: the Max-Flow Problem. The effectiveness of the reduction is due to the fact that it allows to underline the relations among the set of the events and to consider the subset of feasible combinations. The details of the algorithm are omitted here. We simply recall that the method broadly consists of building a Max-Flow problem from the set of events belonging to $E_t$ and, after the max flow is found, the subset $P_{max} \subseteq E_t$ ($P_{min}$), of events that gives the maximum (minimum) value of the resource level at the instant $t$, is computed by collecting all activities that are reachable from the source in the residual graph of the Max-Flow problem. Moreover is not necessary to compute the resource-level envelope all possible instants $t$. Indeed, we only need to compute $L_j^{max}$ at times when either $B_t$ or $E_t$ changes. This can only happen at $lb(t_i)$ or $ub(t_i)$ for any time point $t_i$.

### 4.2.1 Detecting the contention peaks over the resource envelope

Once the Resource Envelope is computed it can be used to identify the current *contention peaks* and the sets of activities related to them. A contention peak is a set of temporally overlapping activities which require an amount of a resource greater than its availability.

A first method [Policella et al., 2003] to collect such peaks consists in the following steps: computing the resource envelope profile, detecting the intervals of over allocation, collecting the set of activities involved in such an interval (peak). The latter step entails picking the set of activities that can be executed in such an interval. Unfortunately this approach has the drawback of producing sets of activities that can be greater than necessary. For example let's consider a problem with three activities $a_1$, $a_2$ and $a_3$ with the same interval of allocation and the precedence $a_1\{before\}a_2$. In such a case the method collects the peak $\{a_1, a_2, a_3\}$ meanwhile the better method should collect the two peaks $\{a_1, a_3\}$ and $\{a_2, a_3\}$.

A more careful method should avoid such an aliasing effect. In particular a better method consists in considering the set $P_{max}$ described above. This method is based on the particular case that each activity only uses the several resource without production and/or consumption. Whether the value of the resource envelope in $t$ is greater than the resource capacity, $L^j_{max}(t) \geqslant max_j$, the contention peak will be composed by all the activity $a_i$ such that the time point associated to its start time is in $P_{max}$ but not the time point associated to its end time, that is:

$$contention\ peak = \{a_i | t_{2i-1} \in P_{max} \wedge t_{2i} \notin P_{max}\}$$

For avoiding to collect a set of contention peak redundant the extraction of the contention peak will be done only if exists at least one end time of an activity $a_i$, $t_{2i}$, such that it moves from $A_{t-1}$ to $B_t \cup E_t$.

### 4.2.2 Incremental Computation of the resource envelope

Using a Max Flow algorithm at each step for computing the resource envelope can take much time. For this reason in this section we complete the resource envelope method describing some properties in the evolution of the partition $\mathcal{T} = B_t \oplus E_t \oplus A_t$ and of the set $P_{max}$ described before. These will help us to reduce the impact of using the resource envelope, thus a Max Flow algorithm.

A necessary concept for the theorem below is the concept of contribution of a time point $t_i$:

Definition 4.3: Given a time point $t_i$ and a resource $r_j$ we define as its overall contribution the value:

$$\Sigma ru_{ij} = ru_{ij} + \sum_{\forall t_k | d_{ik} < 0} ru_{kj}$$

Theorem 4.1: If exists a time point $t_i \in E_t \cap E_{t+1}$ and $t_i \in P_{max}(E_t)$, then $t_i \in P_{max}(E_{t+1})$.

**Proof**: *By absurd.* If were that $t_i \in P_{max}(E_t)$ and $t_i \notin P_{max}(E_{t+1})$, it would means that in $t+1$ the contribution of the time point $t_i$ is negative. To be negative must exist a time point $t_k$, $ru_{kj} > -\Sigma ru_{ij}$ and $t_k \in E_{t+1} \cap O_t$ and $st(t_k) < st(t_i)$. But the last two formula are inconsistent each other, indeed if $st(t_k) < st(t_i)$ then $t_k \in C_t \cup E_t$ that disagrees with $t_k \in O_t$.

From this theorem it derives that at each instant $t$ we need to consider only the events in $E_t - P_{max}$ to figure out what events to insert into $P_{max}$. Moreover, from the previous theorem, we can prove the following corollary:

Corollary 4.1: If $E_{t+1} \backslash P_{max}(E_t) = E_t \backslash P_{max}(E_t)$ then $P_{max}(E_{t+1}) = P_{max}(E_t)$.

Unfortunately, for what concerns the events that belong to $E_t$ and $E_{t+1}$ but not to $P_{max}(E_t)$ in $t$ we can claim nothing. Anyway we can prove the following necessary condition:

Theorem 4.2: A possible candidate to $P_{max}$ exists only if it exists at least either a production event in $t_i^+$, $ru_{ij} > 0$, such that $t_i^+ \in O_t \cap E_{t+1} \cap B_{t+1}$ or one consumption event in $t_i^-$, $ru_{ij} < 0$, such that $t_i^- \in (E_t \backslash P_{max}(E_t)) \cap C_{t+1}$.

**Proof**: We prove the two cases separately *Case 1*: if $t_i \in O_t \cap E_{t+1} \cap B_{t+1}$ then a further element is added to $P_{max}$ only if $ru_{ij} > 0$. Indeed if $ru_{ij} \leqslant 0$ it means that it exists at least one production $t_k^+$ that is implied by $t_i^-$. Then would be possible to put only $t_k^+$ in the set $P_{max}$ having a bigger value of $L_{max}$. Then $ru_{ij} > 0$.
*Case 2*: if it exists $t_i \in (E_t \backslash P_{max}(E_t)) \cap C_{t+1}$ then a further element is added to $P_{max}$ only if $ru_{ij} < 0$. Indeed if $ru_{ij} > 0$ then it would exist a time point $t_k$ s.t. its contribute $\Sigma ru_{kj} > 0$ and the combined contribute of $t_i$ and $t_k$ is negative. But this is possible only if $\Sigma ru_{ij<0}$ that is at least a time point $t_z \in (E_t \backslash P_{max}(E_t)) \cap C_{t+1}$ s.t. $ru_{zj} < 0$.

The theorems above allow to reduce the number of times that is necessary to recompute the set $P_{max}$, Theorem 4.2, and the size of set from which extract it, from $E_{t+1}$ to $E_{t+1} \backslash P_{max}(E_t)$.

## 4.3 The Earliest Start Time Approach

The fundamental 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 "real" problems/conflicts in a particular situation (earliest start time solution). In other words the first approach says what *can* happen in such a situation relative to the entire set of possible solutions, the second one, instead, what *will* happen in such a particular case.

The drawback of this approach is that it insures that it exists only one solution of the problem, the earliest start time solution. Using a PCP manner to compute the solution this will consist in a partial ordered among the activities that is in a set of temporal solution $S_T$, but in general this is not a set of solutions of the problem, that is $S_T \nsubseteq \mathcal{S}$. The following section describes a method to generalize this set of temporal solutions in a partial ordered schedule, $\mathcal{POS}$.

### 4.3.1 Producing flexible solutions

In [Cesta et al., 1998] the authors suggest an approach for translating a fixed schedule to a MCM-SP problem instance into a flexible solution. The MCM-SP problem involves a set of activities $a_i$, each of them requiring only the use of a single resource for its entire duration. Given a solution the transforming method, named *chaining*, consists of creating sets of chains of activities, one set for each resource. This operation is accomplished by deleting all previously posted leveling constraints and using the solution resource profiles to post a new set of constraints. In this section we generalize

**Algorithm 3** Chaining($\mathcal{P}, \mathcal{S}$)

**Input:** A problem $\mathcal{P}$ and a its solution $\mathcal{S}$
**Output:** A partial order schedule $\mathcal{S}^*$

```
S* ← P
Sort all the activities according to their earliest start time in S
for all activity a_i do
  for all resource r_j do
    k = 1
    while ru_ij > 0 do
      if queue_jk ≠ ∅ then
        (a, t_1, t_2) ← LastElement(queue_jk)
        if est_{a_i} ⩾ t_2 then
          Enqueue(queue_jk, a_i)
          S* = S* ∪ {a{before}a_i}
          ru_ij ← ru_ij − 1
        end if
      else
        Enqueue(queue_jk, a_i)
        ru_ij ← ru_ij − 1
      end if
      k ← k + 1
    end while
  end for
end for
```

that method for problems that involve multi-capacited resources. This requires a few adjustments:

- a first step is to consider a resource $r_j$ with capacity $c_j$ as a set $R_j$ of $n = c_j$ single capacity sub-resources. The idea is to create a similar situation to the MCM-SP case;
- in this light the second step is to ensure that each activity is allocated to the same subset of $R_j$. This step can be viewed in Figure 3: on the left there is the resource profile of a resource $r_j$, each activity is represented with a different color. The second step consists of maintaining the same subset of sub-resources for each activity over time. For instance, in the center of Figure 3 the light gray activities is re-drawn in the way that it is always allocated on the fourth sub-resource;
- the last step is to build a chain for each sub resource in $R_j$. On the right of Figure 3 this step is represented by the added constraints. That 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, indeed, to avoid this problem, taking into account the different allocation on the set of sub-resources $R_j$.

Figure 3 contains the sketch of a chaining algorithm. It uses a set of queues, $queue_{jk}$, to represent each capacity unit of the resource $r_j$. The algorithm starts by sorting the

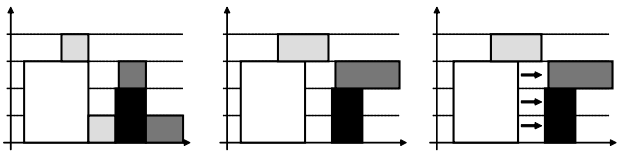

Fig. 3: Chaining method: intuition

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 on a queue a new constraint between this activity and the previous in the queue is posted.

To collect the contention peaks over the earliest start time profile a first approach [Policella et al., 2003] equates the detection of the intervals of over allocation and the collection of the set of activities involved in such intervals. This method can lead to collect a redundant set of contention peaks.

As in the case of the resource envelope analysis in the particular case of activities that only use the 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 that is, sets of activities such that none of the sets is a subset of the others (see [Cesta et al., 2002] for a detailed description).

## 4.4 Synthesis, analysis and resolution of conflicts

What is needed to configure a complete search procedure are mechanisms and heuristics for recognizing, prioritizing and resolving conflicts. This phase can be subdivided in three steps: in the first step the conflict peak to solve is selected. Then, inside the selected contention peak will be necessary to identify the possible pairs $\langle a_i, a_j \rangle$ of activities, or *conflict*, to order for solving the fault. Last, these conflicts will be analyzed selecting one conflict according to the most constrained first principle. Given the pair of activities the order between them will be chosen according to a least constraining principle. The idea in this last point is to resolve the conflict that are the most "dangerous" and solve it with a commitment as smaller as possible.

For evaluating the contention peaks we have used the heuristic estimator provided in [Laborie and Ghallab, 1995]. Given a contention peak and a set of possible ordering constraints $\{oc_1, \ldots, oc_k\}$ which can be posted between pairs of the activities the estimator $K()$ is defined:

$$\frac{1}{K()} = \sum_{i=1}^{k} \frac{1}{1 + commit(oc_i) - commit(oc_{min})} \tag{4}$$

where $commit(oc_i)$ estimates the loss in temporal flexibility. The heuristic estimator $K()$ chooses the contention peak with with highest value. The conflict resolution heuristic simply chooses $oc_{min}$.

As it is underlined in [Smith and Cheng, 1993], given a pair $\langle a_i, a_j \rangle$ of activities in a given contention peak, four possible conditions can held between the two activities according to the maximum distance, $d()$, between two events:

condition 1 : $d(e_{a_i}, s_{a_j}) < 0 \wedge d(e_{a_j}, s_{a_i}) < 0$. In this case there is no way to order the activities. This is identified as a *pairwise unresolvable conflict*.

condition 2 : $d(e_{a_i}, s_{a_j}) < 0 \wedge d(e_{a_j}, s_{a_i}) \geq 0 \wedge d(s_{a_i}, e_{a_j}) > 0$. There is only one feasible ordering the two activities $a_j\{before\}a_i$.

condition 3 : $d(e_{a_i}, s_{a_j}) \geq 0 \wedge d(e_{a_j}, s_{a_i}) < 0 \wedge d(s_{a_j}, e_{a_i}) > 0$. Like the previous one this is also a *pairwise uniquely resolvable conflict*. In this case the relation is $a_i\{before\}a_j$.

condition 4 : $d(e_{a_i}, s_{a_j}) \geq 0 \wedge d(e_{a_j}, s_{a_i}) \geq 0$. In this case we have a *pairwise resolvable conflict*. Both orderings $a_i\{before\}a_j$ and $a_j\{before\}a_i$ are feasible and a choice is needed.

This procedure returns the ordering constraint that leaves the most temporal flexibility: $a_i\{before\}a_j$ whether $d(e_{a_i}, s_{a_j}) > d(e_{a_j}, s_{a_i})$ and $a_j\{before\}a_i$ otherwise. As the reader can see, decisions are taken according to a least commitment principle, trying to retain the maximum amount of temporal flexibility. For that reason the values of the distances $d(e_{a_i}, s_{a_j})$ and $d(e_{a_j}, s_{a_i})$ have a key role.

To consider each pair of activities in each conflict peak may lead to an over commitment. For instance let's consider a resource $r_j$ with capacity $max_j = 3$ and three activities $a_1$, $a_2$ and $a_3$ conflicting for this resource. Each activity requires respectively 1, 2 and 3 units of the resource. Considering all possible pair of activities will lead to take into account the pair $\langle a_1, a_2 \rangle$. This pair will not resolve the conflict because the combined capacity requirement is equals to the capacity. An enhanced procedure, to avoid such a problem, is based on the identification of *Minimal Critical Sets* inside each contention peak. A *Minimal Critical Set*, MCS, is *a set of activities that simultaneously requires a resource $r_j$ with a combined capacity requirement greater than its capacity $c_i$ and the requirement of any subset is lower than, or equals to $c_i$* [Laborie and Ghallab, 1995]. Application of this method can be seen generally as a filtering step. It extracts from each contention peak those sub-sets of activities that are necessary to solve. Unfortunately, as a matter of fact the high computational complexity of enumerating all MCSs prohibits its use on scheduling problems of any interesting size. In [Cesta et al., 2002] two methods to overcome such a problem are described. They consist of sampling a subset of the set of all MCSs.

Linear sampling. A queue $Q$ is used to select an MCS from a contention peak P. Activities $a_i$ are considered sequentially and inserted in $Q$ until the sum of the resource requirement is greater than the resource availability. Then the set $Q$ is saved in a list of MCS and the first element in $Q$ is removed. The previous steps are iterated until there are no more activities.

Quadratic sampling. This is an extension of the previous schema in which the second step is expanded as follows. Once the correct MCS has been collected, instead of removing the first element from $Q$ a forward search through the remaining activities is performed to collect all MCS that can be obtained by dropping the last item placed in $Q$ and substituting with single subsequent activities until an MCS is composed.

These two methods, MCS linear and MCSquadratic, have been used to extract MCSs from the set of conflict peak. These will be analyzed by the $K()$ estimator.

The combination of these three methods with the two different ways of maintaining resource information leads to six different approaches: 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

| | sol. (%) | makespan | CPU-time(sec.) | constraints |
|---|---|---|---|---|
| J10 | 77.04 | 58.31 | 0.32 | 11.54 |
| J20 | 50.74 | 96.48 | 3.88 | 33.40 |
| J30 | 43.33 | 118.17 | 24.47 | 63.29 |

Tab. 1: EBA

| | sol. (%) | makespan | CPU-time(sec.) | constraints |
|---|---|---|---|---|
| J10 | 96.30 | 47.35 | 0.32 | 6.40 |
| J20 | 95.56 | 72.90 | 1.75 | 18.69 |
| J30 | 96.30 | 79.21 | 5.40 | 35.10 |

Tab. 2: ESTA$^C$

section presents a discussion of the results obtained testing the six approaches on a significant problem scheduling benchmark: RPCSP/max.

## 4.5 Empirical evaluation

For the evaluation we consider the Resource-Constrained Project Scheduling Problem with Minimum and Maximum time lags (RPCSP/max), which involves synchronizing the use of a set of renewable resources $R = \{r_1 \dots r_m\}$ to perform a set of activities $V = \{a_1 \dots a_n\}$ over time. The execution of each activity is subject to the following constraints:

- each activity $a_j$ has a duration $dur_{a_j}$, a start time $s_{a_j}$ and an end time $e_{a_j}$ such that $e_{a_j} = s_{a_j} + dur_{a_j}$;
- each activity $a_i$ requires the use of $ru_{ik}$ units of the resource $r_k$.
- a set of temporal constraints $c_k$ defined for an activities pair $(a_i, a_j)$ of the form of $c_k^{min} \leq s_{a_j} - s_{a_i} \leq c_k^{max}$;
- each resource $r_k$ has an integer capacity $cap_k \geq 1$;

A solution to a RCPSP/max is any consistent assignment to the start-time of all the activities in $V$ which does not violate resource capacity constraints.

The results we will show have been obtained using the benchmarks defined in [Kolisch et al., 1998]. These consist of three sets $J10$ $J20$ and $J30$ of 270 instances of different size $10 \times 5$, $20 \times 5$ and $30 \times 5$ where the numbers represent respectively the number of activities and of resources involved. All algorithms presented in this paper are implemented in C++ and the CPU times presented in the following tables are obtained on a Pentium 4-1500 Mhz processor under Windows XP.

An initial comparison is presented according to the following parameters: (1) percentage of problems solved from a fixed set, (2) average CPU-time spent to solve instances of the problem, (3) average makespan and (4) the number of leveling constraints posted to solve the problem (or search steps). The latter gives the average number of search steps required to achieve a solution. We also consider the makespan because it gives the quality of the solution in the best case (no disruptions) possible. Later, in Section 4.5.1, we analyze the flexibility of the solutions achieved using each different

| | sol. (%) | makespan | CPU-time(sec.) | constraints |
|---|---|---|---|---|
| J10 | 85.19 | 55.29 | 0.77 | 11.12 |
| J20 | 71.11 | 92.65 | 11.35 | 32.87 |
| J30 | 68.89 | 112.14 | 48.89 | 56.84 |

Tab. 3: EBA + MCS linear sampling

| | sol. (%) | makespan | CPU-time(sec.) | constraints |
|---|---|---|---|---|
| J10 | 97.78 | 55.47 | 0.91 | 12.38 |
| J20 | 89.63 | 94.03 | 13.21 | 34.98 |
| J30 | 82.22 | 116.10 | 68.22 | 59.64 |

Tab. 4: EBA + MCS quadratic sampling

approach. The parameters introduced in Sect. 3.2 will be used as a basis for such an evaluation.

In Table 1 the results of the resource envelope-based approach without MCS filtering, EBA, are shown. Comparing these values with those obtained with ESTA$^C$, Table 2, it can be seen that the EBA approach is actually quite ineffective. It solves significantly fewer problems than ESTA$^C$ in each problem set and incurs higher CPU times.

A significant drawback of using the resource envelope is its high associated computational cost. Indeed, the computation of the envelope implies that it is necessary to solve a Max-Flow problem for each time-point. As indicated in [Muscettola, 2002], this leads to an overall complexity of $O(n^4)$ which can be reduced to $O(n^{2.5})$ in practical cases. These computational requirements present a formidable barrier to effective application of the resource envelope. In the current implementation we use a Max Flow method based on the *pre-flow* concept (for details see [Goldberg and Tarjan, 1988] and [Cherkassky and Goldberg, 1997]. The use of the properties described in the Sect 4.2.2 speeds up the solving process avoiding to require a re-computation of the envelope at each step of the search. Moreover theorem 4.2 allows to apply the Max Flow algorithm on a subset of $E_t$: $E_t \backslash P_{max}(E_{t-1})$.

Comparison with the results obtained with the ESTA$^C$ algorithm highlight a further negative aspect of the EBA approach: EBA consistently requires a bigger number of search steps than ESTA$^C$ in generating a solution. In fact, this result could have been predicted. The ESTA$^C$ approach, indeed, posts a set of "implicit" constraints each time the profile is computed: each activity has to start at its earliest start-time. These constraints temporarily restrict the solution's temporal flexibility (for the purpose of computing resource profiles). This avoids having to take into account all the possible temporal configurations of the set of the activities, and it follows that a smaller set of constraints are necessary to order them.

By adding MCS filtering to the EBA search configuration, we obtain a noticeable improvement of the result. Tables 3 and 4 represent respectively the results obtained using the linear and the quadratic sampling versions of MCS filtering. The use of MCS linear sampling gives an overall improvement of the basic approach. Although the results are still worse than the ESTA$^C$ case, the MCS linear gives better values than the simple EBA with respect to all performance parameters. The use of the quadratic MCS version gives considerable further improvement (Table 4) with respect to the number of problems solved, and indeed the performance along this dimension is closer to that

| | sol. (%) | makespan | CPU-time(sec.) | constraints |
|---|---|---|---|---|
| J10 | 98.15 | 46.63 | 0.34 | 6.23 |
| J20 | 96.67 | 72.45 | 2.14 | 17.49 |
| J30 | 96.67 | 78.45 | 8.08 | 34.07 |

Tab. 5: ESTA$^C$+MCS linear sampling

| | sol. (%) | makespan | CPU-time(sec.) | constraints |
|---|---|---|---|---|
| J10 | 98.15 | 46.70 | 0.34 | 6.26 |
| J20 | 96.67 | 72.75 | 2.27 | 17.70 |
| J30 | 97.04 | 78.55 | 9.51 | 34 |

Tab. 6: ESTA$^C$+MCS quadratic sampling

achieved with the ESTA$^C$ approach. This could be predicted from the fact that the quadratic version takes a larger sample than the linear version from the space of the MCS. On the other hand, EBA with quadratic MCS has a negative impact on the CPU-time. This aspect follows from use of the more expensive MCS quadratic sampling procedure in conjunction with the resource envelope computation. Finally, Tables 5 and 6 present the results obtained using the two MCS sampling methods in conjunction with the earliest start time approach. Here we see only slight improvement over the basic ESTA$^C$ procedure.

### 4.5.1 Flexibility

This section analyzes the quality of the solutions obtained using either the resource envelope or the earliest start time profile to guide the algorithm.

To normalize the values obtained with respect to the complexity of the problem solved we present for each parameter $\mu()$ the following value:

$$\Delta\mu(\mathcal{P},\mathcal{S}) = \frac{\mu(\mathcal{P}) - \mu(\mathcal{S})}{\mu(\mathcal{P})} \times 100$$

where $\mathcal{P}$ and $\mathcal{S}$ are the value of the parameter for respectively the problem and its solution. The value $\Delta\mu(\mathcal{P},\mathcal{S})$, then, equates to the cut, in terms of quality, that the solving process introduces. Moreover taking into account that different sets of problems were solved by each approach we only consider the subset of problem instances solved by all six approaches. Table 7 presents the results of the six different approaches according to the three parameters described in Sect. 3.2.

In the current evaluation we consider, for the metric (3), that the probability that a disruption occurs during the execution of an activity $a_i$ is related to its duration and to the overall duration of the solution ($mk$), $Pr_{disr}(a_i) = \frac{dur_{a_i}}{mk}$. Furthermore for computing the number of changes we assume the biggest shift $\Delta_i$ possible for activity $a_i$ in the worst case, that is, $num_{changes}(a_i, let(a_i) - eet(a_i))$. Then we can re-write the (3) as:

$$dsrp = \frac{1}{N}\sum_{i=1}^{N} \frac{dur_{a_i}}{mk} \times \frac{let(a_i) - eet(a_i)}{num_{changes}(a_i, let(a_i) - eet(a_i))} \quad (5)$$

| | $\Delta flex$ | | | $\Delta fldt$ | | | $\Delta dsrp$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | J10 | J20 | J30 | J10 | J20 | J30 | J10 | J20 | J30 |
| EBA | 86.27 | 83.77 | 76.80 | 37.21 | 35.94 | 30.77 | 46.89 | 42.09 | 41.63 |
| EBA+MCS lin. | 83.36 | 84.94 | 81.57 | 35.11 | 39.57 | 41.41 | 44.37 | 41.49 | 42.78 |
| EBA+MCS quad. | 83.99 | 86.54 | 83.58 | 35.20 | 41.81 | 44.33 | 44.90 | 43.23 | 43.56 |
| ESTA$^C$ | 80.56 | 79.96 | 74.98 | 32.79 | 35.27 | 40.79 | 35.96 | 25.99 | 27.17 |
| ESTA$^C$+MCS lin. | 79.79 | 80.41 | 74.97 | 32.42 | 34.87 | 40.94 | 34.75 | 26.74 | 28.39 |
| ESTA$^C$+MCS quad. | 79.94 | 80.79 | 75.26 | 32.46 | 35.56 | 39.61 | 35.16 | 28.37 | 27.55 |

Tab. 7: $\Delta\mu(\mathcal{P}, \mathcal{S})$ for the three metrics: $flex_{seq}$, $fldt$ and $dsrp$.

At the first sight it is clear, from the Table 7, that the ESTA$^C$ approaches, along the three dimensions (metrics), dominate the EBA approaches across all problem sets. We recall that the values in the table represent lack of quality that the different methods introduce. A particular aspect is the analysis of the use of MCS filtering methods. We can see that in general they introduce a slight lost of quality. Then using the MCS filtering methods we notice a trade off between the effectiveness and the efficiency.

**Final remarks** Considering the philosophies behind the different approaches that have been evaluated, one would expect that those that manage the knowledge of all the possible temporal allocations would provide the most effective basis for generating flexible solutions. As a matter of fact, we have shown a two step procedure for computing a first schedule and translating it into a $\mathcal{POS}$ solution to be a more effective approach. The first step allows advantage to be taken of the effectiveness of the ESTA approach (i.e., makespan and CPU time minimization), while the second step, has been shown to be capable of re-instating temporal flexibility in a way that preserves the qualities of the initial solution.

## 5 Open Issues

To conclude in this section we summarize some possible future directions, identifying which are necessary to explore for completing the current work and which, according to the time available, will be useful to analyze for enriching the final work. In the first set we have identified the following extensions:

Simulation A fundamental direction toward developing the current work concerns the analysis of a simulation of the execution of the solutions achieved. This point might involve several aspects, like the project of a simulated environment, the sources of uncertainty, the parameters for monitoring the execution and techniques for managing the solution. Indeed, as for the last point, the approach we are pursuing, insures a set of solution for which it is possible to switch rapidly (propagating the new disturbances). Unfortunately it is possible to meet changes which are not provided for the set of solutions available. In this case a repair approach will be necessary.

A complete search algorithm As we have said before at this stage of the work we limit to use greedy algorithms because we aimed at understanding the phenomena behind the approaches implemented. In the future will be necessary to use a more complete search framework. This might allow to increase the number of problem solved overcoming the lack of guidance we can have using a greedy algorithm.

Optimization This point is related with the previous one. On the basis of the three metrics introduced during the Sect. 3.2, it is possible to think about the implementation of optimization method. As the reader could see the current methods have not such a goal they only try to find a solution, $\mathcal{POS}$, for the problem. For instance using a complete search algorithm we can optimize one of the three parameters or one their combination. Another possible approach in the case of ESTA$^C$, might be the use of a more sophisticated chaining operator like a local search algorithm.

Enrich the concept of partial order schedule A further goal is to extend the concept of partial order schedule to a larger set of possible kind of external disturbances. Indeed in the current work we narrow our attention on possible temporal variation, like greater duration of an activity and delay in the start of an activity. We would like to extend our analysis to hedge against further disruptions like resource breakdown and/or new task to be executed.

Once the previous aims will be achieved we will have further directions to investigate. Among these we would like to underline the following:

Applying Constraint Propagation The approach realized is based on the constraint satisfaction problem paradigm. An important tool of this model is the constraint propagation. The current implementation uses only the temporal propagation, that is, new constraints are deducted through the time constraints in the situation. Moreover the polynomial complexity of temporal propagation insures the fast reaction of a $\mathcal{POS}$ to external events. On a different perspective is possible to use constraint propagation rules to improve the approaches realized. For instance we are inquiring about the use of resource propagation (see [Laborie, 2003]). They seem, in particular for the resource envelope approach, complementary with the current solving process. We are expecting that they, pruning the search space, should support the decision phase, increasing the global effectiveness.

New problem scheduling Even though the kind of scheduling problem we are facing in the current work is very significant further steps can be done in this direction. An important aspect in the RCSPS/max problem is the presence of time-window constraints. These are very interesting especially in the execution phase because they might avoid to shift forward activities. This type of problem does not include some aspects like production and/or consumption activities and, more generally, activities that use the resource in a no step-wise way.

## 6 Summary

The aim of our work is to build scheduling solutions able to hedge against unexpected events. We focus our attention to temporal disruptions: activity duration lasts more than expected, the start time of an activity is shifted forward. For such a case we define a particular robust schedule, partial order schedule, $\mathcal{POS}$. Moreover we have investigated two orthogonal approaches: EBA and ESTA$^C$.

For evaluating the quality of a solution we introduce a set of parameters (Sect. 3.2) which are used for testing the two approaches implemented (Sect. 4). The two approaches are base on two different method for maintaining profile information: considering all the temporal solutions (resource envelope) and analyzing the profile for a precise temporal solution (earliest start time solution).

## Acknowledgment

My work was supported by a Scholarship from CNR on Information Technology, and is currently supported by ASI (Italian Space Agency) under project ARISCOM (Contract I/R/215/02). This work has been partially developed during my visit at the CMU Robotics Institute as a visiting student scholar. I would like to thank the members of the Intelligent Coordination and Logistics Laboratory for support and hospitality.

I would also like to express my thanks and gratitude to my advisors Amedeo Cesta and Angelo Oddi, and to Steve Smith, who hosted me at the ICL laboratory, CMU. Their advice and guidance throughout my research work are immeasurable.

## References

[Aloulou and Portmann, 2003] Aloulou, M. and Portmann, M. (2003). An Efficient Proactive Reactive Scheduling Approach to Hedge against Shop Floor Disturbances. In *Proceedings of the $1^{st}$ Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2003*, pages 337–362.

[Cesta et al., 1998] Cesta, A., Oddi, A., and Smith, S. F. (1998). 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*.

[Cesta et al., 2002] Cesta, A., Oddi, A., and Smith, S. F. (2002). A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristic*.

[Cherkassky and Goldberg, 1997] Cherkassky, B. V. and Goldberg, A. V. (1997). On Implementing the PushRelabel Method for the Maximum Flow Problem. *Algorithmica*, 19(4):390–410.

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

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

[El Sakkout and Wallace, 2000] El Sakkout, H. and Wallace, M. (2000). Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4). Special Issue on Industrial Constraint-Directed Scheduling.

[Ginsberg et al., 1998] Ginsberg, M., Parkes, A., and Roy, A. (1998). Supermodels and Robustness. In *Proceedings of the $15^{th}$ National Conference on Artificial Intelligence, AAAI-98*, pages 334–339. AAAI Press.

[Goldberg and Tarjan, 1988] Goldberg, A. V. and Tarjan, R. E. (1988). A New Approach to the Maximum Flow Problem. *Journal of the Association for Computing Machinery*, 35(4):921–940.

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

[Kumar, 2003] Kumar, T. K. S. (2003). Incremental Computation of Resource-Envelopes in Producer Consumer Models. In *Principles and Practice of Constraint Programming, $9^{th}$ International Conference, CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 664–678. Springer.

[Laborie, 2003] Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2):151–188.

[Laborie and Ghallab, 1995] Laborie, P. and Ghallab, M. (1995). Planning with Sharable Resource Constraints. In *Proceedings of the $14^{th}$ International Joint Conference on Artificial Intelligence, IJCAI-95*, pages 1643–1649.

[Muscettola, 2002] Muscettola, N. (2002). 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.

[Policella et al., 2003] Policella, N., Smith, S. F., Cesta, A., and Oddi, A. (2003). Steps toward Computing Flexible Schedules. In *Proceedings of Online-2003, Online Constraint Solving: Handling Change and Uncertainty*, Kinsale, Co. Cork, Ireland.

[Smith, 1994] Smith, S. F. (1994). OPIS: A Methodology and Architecture for Reactive Scheduling. In Fox, M. and Zweben, M., editors, *Intelligent Scheduling*. Morgan Kaufmann.

[Smith and Cheng, 1993] Smith, S. F. and Cheng, C. (1993). 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.