Steps toward Computing Flexible Schedules

Nicola Policella[1⋆,⋆⋆], Stephen F. Smith[2], Amedeo Cesta[1], and Angelo Oddi[1]

[1] Planning & Scheduling Team, Institute for Cognitive Science and Technology
Italian National Research Council
Rome, Italy
{policella,cesta,oddi}@ip.rm.cnr.it
[2] The Robotics Institute, Carnegie Mellon University
Pittsburgh, PA, USA
sfs@cs.cmu.edu

Abstract. In this paper we consider the problem of building schedules that retain temporal flexibility. Such a feature represents a relevant benefit for managing changes in a dynamic environment. We begin by formalizing the concept of flexibility, to provide a set of metrics against which the flexibility of competing schedules can be compared. Then, using a common solving framework, we develop two orthogonal procedures for constructing a flexible schedule. The first, which we call the resource envelope based approach, uses computed bounds on cumulative resource usage (i.e., a resource envelope) to identify potential resource conflicts, and progressively winnows the total set of temporally feasible solutions into a smaller set of resource feasible solutions by resolving detected conflicts. The second, referred to as the earliest start time approach, instead uses conflict analysis of a specific (i.e., earliest start time) solution to generate an initial fixed-time schedule, and then generalizes this solution to a set of resource feasible solutions. We evaluate the relative effectiveness of these two procedures on a set of project scheduling benchmark problems, considering both their problem solving performance and the flexibility of the solutions they generate.

1 Introduction

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.

In this paper we pursue 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. Our particular focus is generation of schedules that retain temporal

⋆ Ph.D. student at the Department of Computer and Systems Science, University of Rome "La Sapienza", Italy

⋆⋆ Visiting student scholar at the Robotics Institute, Carnegie Mellon University

flexibility. Historically, a major obstacle to generating temporally flexible schedules has been the difficulty of accurately computing the number of resources required across all possible executions. Without this capability, it is difficult to obtain sufficient search guidance to achieve scalable problem solving performance. In [1], this problem is circumvented through use of a two-step procedure, where attention is first focused on generating a particular resource-feasible solution, the earliest start time solution, and then this solution is generalized into a flexible solution. However, in [6], a new procedure for computing resource usage bounds for a flexible schedule has been proposed, referred to as the *resource envelope*. Since this procedure generates "the tightest possible resource-level bound for a flexible plan", it suggests the possibility of generating a flexible schedule in a more direct, least-commitment fashion, using the resource envelope to detect potential resource conflicts and transforming the set of possible solutions into a small set of resource-feasible solutions by successively posting new conflict-resolving constraints between competing activities. Intuitively, we might expect that a schedule generation scheme which operates in such a least-commitment fashion would produce a solution with greater flexibility than an approach that instead produces a single point solution and attempts to generalize from this. But the performance tradeoffs are not immediately clear. To investigate these tradeoffs, we develop concrete implementations of each of the above approaches. To sharpen the comparison, we utilize a common scheduling framework wherein schedule generation is formulated as an incremental, conflict removal (or leveling) process. We experimentally compare the performance of each procedure on a set of known resource-constrained project scheduling problems, considering both problem solving and solution flexibility characteristics.

The paper starts by discussing the concept and benefits of flexible solutions in uncertain environments and specifying two parameters for measuring solution quality along this dimension (Sect. 2.1). In Sect. 3 we describe the precedence constraint posting (PCP) framework in which the two schedule generation approaches are to be defined and compared. In Sect. 4 and Sect. 5 the resource envelope and the earliest start time approach are respectively introduced. Section 5.1 then presents a method for obtaining flexible solutions from fixed time ones. An empirical evaluation is presented in Sect. 6, analyzing the feature of flexibility in Sect. 6.1. Finally we summarize our main results.

2 Flexibility and the Uncertainty in Scheduling

In the realm of scheduling problems different sources of uncertainty can arise: durations may not be known, resources may have lower capacity than expexted (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 [10] or the completeness of the new solution [8].

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 [3]. 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 [4]. 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. The aim is to produce solutions that enable reaction to exogenous events without *large changes* or *explicit assistance* (or repairs). A similar concept of producing solutions that promote bounded, localized recovery from execution failures is also proposed in [5]. 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.

2.1 Evaluation criteria

A fundamental point related to the flexibility concept introduced above is the need for metrics that characterize the quality of a flexible solution, and in general, the extent to which a solution is suitable for the execution phase. Different concepts can be used to describe the behavior of a given system facing uncertainty in the world - stability, flexibility or robustness - but these notions remain vague unless both the perturbations and features of interest are specified. Indeed it makes no sense to define the quality of a system without first specifying which of its characteristics have been considered. In the following we represent the scheduling problem by a graph where for each activity a_i there are two nodes (events), the start time s_{a_i} and the end time e_{a_i} and for each constraint there is an edge in the graph. Applying Dijkstra's Shortest Path Algorithm the earliest and latest values for both events of each activity are computed: $est(a_i)$, $lst(a_i)$ $eet(a_i)$ and $let(a_i)$.

In [1] a metric[3] based on the temporal slack associated with each activity is introduced:

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

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 new parameter that takes into account the impact of disruptions on the schedule, or *disruptibility* of a solution:

$$dsrp = \frac{1}{N} \sum_{i=1}^{N} Pr_{disr}(a_i) \times \frac{let(a_i) - eet(a_i)}{num_{changes}(a_i, \Delta_{a_1})} \quad (2)$$

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 Δ_{a_i} of the activity a_i is computed. In Sect. 6.1 both the probability distribution, $Pr_{disr}(a_i)$, and the right shift, Δ_{a_i}, used in the empirical evaluation are described.

The intuition behind this 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.

3 A Precedence Constraint Posting Framework

The goal of the paper is to evaluate the ability to find flexible solutions using two different methods to estimate the resource needs at each instant: the earliest start time profile [1] and the resource envelope [6]. For performing such a comparison we will use each to guide the search within a profile-based scheduling framework. Within this framework, a resource feasible 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. There are different ways of representing and maintaining profile information. We will compare the extreme ones: the resource envelope, which maintains all possible solutions, and the earliest start time approach, which considers a single solution. In the latter case, as

[3] Named *rb*, robustness, in [1].

PCP-greedy(*Problem*)
Input: A problem
Output: A conflict-free solution
1. *CurrentSituation* ← *Problem*
2. **if** Exists-Unresolvable-Conflict(*CurrentSituation*)
3. **return** NIL
4. **else**
5. *ConflictSet* ← Select-Conflict-Set(*CurrentSituation*)
6. if *ConflictSet* = ∅
7. **return** *CurrentSituation*
8. **else**
9. *constraint* ←Select-Leveling-Constraint(*ConflictSet*)
10. Add-Constraint(*CurrentSituation*, *constraint*)
11. PCP-greedy(*CurrentSituation*)

Fig. 1. Conflict-free Algorithm

it finds a single solution, a robust solution will be built as a two step process of finding a fixed-time solution and then generalizing from it as current constraints will permit.

The framework is based on a constraint satisfaction model of scheduling problems in which each activity a_i is represented by two events, the start time s_{a_i} and the end time e_{a_i}. There are two aspects to take into account: the time and the resource constraints. The former introduces a set of constraints that represent either the duration of the activity, $dur_{a_i}^{min} \leq e_{a_i} - s_{a_i} \leq dur_{a_i}^{max}$, or the relation between a pair of activities, $c_{ij}^{min} \leq s_{a_j} - s_{a_i} \leq c_{ij}^{max}$. Representing the resources requires taking into account the usage of each resource r_k by the different activities. This is done by associating a resource usage value at each event, $ru_{ik}(tp)$. This allows the representation of different kind of activities: for instance, for an activity a_i that uses ru_{ik} capacity units it will be sufficient to set $ru_{ik}(st_{a_i}) = ru_{ik}$ and $ru_{ik}(et_{a_i}) = -ru_{ik}$. According to previous models the resource constraint for a resource r_k is defined as $\sum_{\forall tp_j \leq t} ru_{ik}(tp_j) \leq cap_k$ for each instant t.

Fig. 1 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 (lines 2–5). This analysis can highlight an infeasible current situation, where resource needs are greater then the availability: *contention peak* (line 6). For solving such a case, a new precedence constraint is synthesized and added to the problem (lines 9-11). What is needed to configure a complete search procedure are mechanisms and heuristics for recognizing, prioritizing and resolving conflicts. These strategies derive from those first introduced in [11] and extended to the cumulative resource case in [1]. A conflict is defined to be any 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.

The previous conditions are used for implementing the following functions utilized in the general schema introduced in Fig. 1:

Exists-Unresolvable-Conflict(*CurrentSituation*). This procedure identifies whether the current situation is infeasible, by propagating the constraints defined in the problem. It detects a contention peak where for each pair of activities condition 1 holds.

Select-Conflict-Set(*CurrentSituation*). This procedure selects a pair $\langle a_i, a_j \rangle$ of activities within a resolvable peak. Two cases are distinguished. When one or more pairwise conflicts satisfy conditions 2 or 3 then the conflict with the minimum (and negative) value $\omega_{res}(a_i, a_j) = \min\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}$ is selected. Alternatively, if condition 4 holds, then is selected the pairwise conflict $\langle a_i, a_j \rangle$ that minimize the value

$$\omega_{res}(a_i, a_j) = \min\{\frac{d(e_{a_i}, s_{a_j})}{\sqrt{S}}, \frac{d(e_{a_j}, s_{a_i})}{\sqrt{S}}\}$$

where $S = \frac{\min\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}{\max\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}$.

Select-Leveling-Constraint(*ConflictSet*). This procedure returns the ordering constraint that leaves the most temporal flexibility: $a_i \leq a_j$ whether $d(e_{a_i}, s_{a_j}) > d(e_{a_j}, s_{a_i})$ and $a_i \geq a_j$ 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 explore the impact of additional heuristic bias on the effectiveness of various instantiations of this greedy search algorithm, we also define an enhanced Select-Conflict-Set procedure which incorporates a further heursitic estimator. Specifically, we add a method for analyzing conflict sets with the aim of avoiding redundant constraints, through identification of *Minimal Critical Sets* [7]. 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*. Application of this method can be seen genrally as a filtering step. It extracts from several conflict sets those sub-sets of activities that are necessary to solve. In [7] a heuristic estimator is also provided. Given a MCS and a set of possible ordering constraints $\{oc_1, \ldots, oc_k\}$ which can be posted between pairs of the activities in MCS the estimator $K(\text{MCS})$ is defined:

$$\frac{1}{K(\text{MCS})} = \sum_{i=1}^{k} \frac{1}{1 + commit(oc_i) - commit(oc_{min})} \tag{3}$$

where $commit(oc_i)$ estimates the loss in temporal flexibility as explained in [7]. As a matter of fact the high computational complexity of enumerating all MCSs prohibits its use on scheduling problems of any interesting size. In [2] 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.

This heuristic estimator leads to a modified Select-Conflict-Set procedure (line 5 of the algorithm in Figure 1): it chooses the MCS with highest K value. The conflict resolution heuristic (Select-Leveling-Constraint) simply chooses oc_{min}.

The next sections introduce the two approaches of interest in this paper for representing and maintaining resource profile information, the resource envelope approach and the earliest start time approach.

4 The Resource Envelope

The first method considered for guiding the search for reaching flexible solutions is the resource envelope defined in [6]. This work proved that 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 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.

To find the maximum (minimum) value of the resource level in an instant t most methods subdivide the set of time points (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

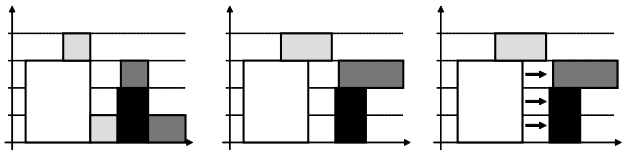

Fig. 2. Chaining method: intuition

developed. In [6], 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. He shows that a polynomial algorithm can be found, taking the relations among the events into account 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. We will discuss the approach obtained using the resource envelope to guide the search (EBA) in Sect. 6.

5 The Earliest Start Time Approach

In [1] it has been shown that use of the earliest start time profile is an effective way to solve scheduling problems. This profile is based on a temporal net property: at each time point (event) tp_i there is an associated interval of possible values $[lb_{tp_i}, ub_{tp_i}]$ and the extremes of the interval if chosen as the value for all time variables tp_i identify a solution of the temporal net. In [1] the earliest start time solution (that is $tp_i = lb_{tp_i}$ for each i) is considered. The method (named ESTA) consists of building the resource profile for such a temporal solution and matching it with the resource bounds. If a violation exists then further constraints are posted to resolve the resource conflict.

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.

As ESTA finds fixed time solutions a method for computing flexible solutions is needed. In the next section we describe a method for achieving such a feature.

Chaining(*Problem, Fixed-Time Solution*)
Input: A problem and an its Fixed-Time solution
Output: A flexible solution
1. $S \leftarrow$ *Fixed-Time Solution*
2. $S^* \leftarrow$ *Problem*
3. Sort all the activities according to their start time in S
4. **For each** activity a_i
5. **For each** resource r_j
6. k=1
7. **While** $ru_{ij} > 0$
8. **If** $Q_{jk} \neq \emptyset$
9. $(a, t_1, t_2) \leftarrow ReadLastElement(Q_{jk})$
10. **If** $est_{a_i} \geq t_2$
11. Add $(a_i, est_{a_i}, eet_{a_i})$ to Q_{jk}
12. Add-Constraint(S^*, $a\{before\}a_i$)
13. $ru_{ij} = ru_{ij} - 1$
14. **else**
15. Add $(a_i, est_{a_i}, eet_{a_i})$ to Q_{jk}
16. $ru_{ij} = ru_{ij} - 1$
17. $k = k + 1$
18. **Return** S^*

Fig. 3. Chaining Algorithm

5.1 Producing flexible solutions

In [1] 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 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 2: 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 2 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 2 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, Q_{jk}, to represent each capacity unit of the resource r_j. The elements of the queues consists of a triple $\langle a_i, est_{a_i}, eet_{a_i} \rangle$, that is, the activity a_i and its start and end time according to the earliest start time solution S. 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 (line 10). Then when an activity is allocated on a queue a new constraint between this activity and the previous in the queue is posted (line 12).

The enhanced algorithm obtained by adding the chaining post processing to the ESTA algorithm has been named ESTA^C. Obviously greater CPU-time is required to use the chaining method, being that it is a post-processing phase. Furthermore using the chaining method two important features of the ESTA approach, the number of solved problems and the makespan, are preserved.

6 Experimental Evaluation

In this section we present the results obtained using either the resource envelope or the earliest start time approach embedded in the common framework introduced in Sect. 3.

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 [9]. These consist of three sets of 270 instances of different size 10×5, 20×5 and 30×5 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 III-500 Mhz processor under Windows NT 4.0.

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

size	sol. (%)	makespan	CPU-time(sec.)	constraints
10	67.4	55.64	4.211	11.15
20	50.7	92.22	120.4	40.26
30	52.6	130.15	1376.4	87.53

Table 1. EBA

size	sol. (%)	makespan	CPU-time(sec.)	constraints
10	97.04	49.1	1.728	7.16
20	95.56	83.5	9.898	21.30
30	95.93	106.1	33.736	38.18

Table 2. ESTA^C

of leveling constraints posted to solve the problem. The last gives one estimate of the kind of flexible solution created (The higher the less desirable). We also consider the makespan because it gives the quality of the solution in the best case (no disruptions) possible. Later, in Section 6.1, we analyze the flexibility of the solutions achieved using each different approach. The parameters introduced in Sect. 2.1 will be used as a basis for that 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 [6], 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 point of fact, use the resource envelope within a scheduling problem solver requires recomputation of the envelope at each step of the search.

Comparison with the results obtained with the ESTA^C algorithm highlight a further negative aspect of the EBA approach: EBA consistently adds a larger set of leveling constraints 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 ba-

size	sol. (%)	makespan	CPU-time(sec.)	constraints
10	76.7	54.83	10.29	10.86
20	69.3	99.88	162.95	30.88
30	66.7	132.7	1432.1	62.84

Table 3. EBA + MCS linear sampling

size	sol. (%)	makespan	CPU-time(sec.)	constraints
10	96.7	57.71	10.78	12.36
20	86.3	106.05	205.76	34.74
30	81.1	143.76	2101.2	65.74

Table 4. EBA + MCS quadratic sampling

sic 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 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 both the CPU-time and the number of leveling constraints added. The first aspect follows from use of the more expensive MCS quadratic sampling procedure in conjunction with the resource envelope computation. The second aspect, instead, as suggested above, is a consequence of the nature of approaches that attempt to retain maximal temporal flexibility. 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.

6.1 Flexibility

This section analyzes the flexibility of the solutions obtained using the resource envelope or the earliest start time profile to guide the algorithm. For each benchmark problem set we present the average value. 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 parameters described in Sect. 2.1.

First consider the metric (1), which reflects the degree of "fluidity" of the generated solutions. The basic EBA approach tends to produce more robust solutions along this dimension when it is able to generate a solution. On the other hand since EBA is managing the total set of possible solutions, search heuristics for conflict selection and resolution generally provide less leverage (see Table 1) than in ESTA^C and there is greater chance of not finding a solution. Furthermore, even as the introduction of MCS filtering increases the percentage of problems solved, it also leads to solutions that on average are less fluid. It thus appears

size	sol. (%)	makespan	CPU-time(sec.)	constraints
10	98.15	48.55	1.832	3.03
20	96.67	82.97	13.37	11.01
30	97.04	106.35	86.220	22.33

Table 5. ESTA^C+MCS linear sampling

size	sol. (%)	makespan	CPU-time(sec.)	constraints
10	98.15	48.58	1.846	3.03
20	96.30	83.08	14.268	10.91
30	97.41	106.01	125.52	22.38

Table 6. ESTA^C+MCS quadratic sampling

that the heuristic bias introduced by MCS filtering has both positive and negative aspects, illustrating the difficult challenge associated with injecting heuristic guidance into the EBA search procedure. This behavior is not observed in the case of ESTA^C: indeed all three algorithms produce solutions with essentially the same fluidity.

To quantify the impact of possible disruption during the execution of the schedule, a second metric (2) was introduced in Sect 2.1. In the current evaluation we consider 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 Δ_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 (2) as:

$$dsrp = \frac{1}{N} \sum_{i=1}^{N} \frac{dur_{a_i}}{mk} \times \frac{let(a_i) - eet(a_i)}{num_{changes}(a_i, let(a_i) - eet(a_i))} \tag{4}$$

Examining the values in the table, we see that along this dimension the ESTA^c approaches dominate the EBA approaches across all problem sets.

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 fixed time schedule and translating it into a flexible solution to be a more effective approach. The first step allows advantage to be taken of the effectiveness of a fixed time scheduling 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 fixed time solution.

	fluidity			disruptibility		
	10	20	30	10	20	30
EBA	28.69	31.32	33.35	8.90	12.74	18.21
EBA+MCS linear	27.05	25.78	25.76	8.74	12.18	17.27
EBA+MCS quadratic	25.84	24.14	22.35	8.23	12.73	18.57
ESTA^C	29.18	29.49	28.09	10.20	16.38	23.80
ESTA^C+MCS linear	29.20	29.97	28.45	10.21	16.49	24.90
ESTA^C+MCS quadratic	29.20	30.05	28.06	10.20	15.34	24.31

Table 7. Fluidity & Disruptibility.

7 Conclusion and Future Work

In this paper, we have investigated two approaches for generating schedules that retain temporal flexibility and possess good robustness properties. Such flexible solutions promote an ability to react to exogenous events with minimal solution change and without external assistance. To support assessment of schedules from this perspective, we first defined measures of solution robustness relating to fluidity and disruptibility. We then developed two alternative approaches to constructing a flexible schedule: one based on use of the resource envelope introduced in [6] (named EBA) and the other based on use of the earliest start time profile [1]. The latter approach also involved the definition of a post processing method to transform a fixed-times schedule into a flexible schedule (the complete approach has been named ESTA^C). To provide a basis for comparative analysis, both approaches were formulated within a common framework.

Analyzing initial performance results obtained with both procedures, we found that EBA was able to solve significantly smaller numbers of problems than ESTA^C at much higher computational cost per solution. To improve EBA's performance, we incorporated two approximate methods for generating Minimal Conflict Sets (MCS): linear and quadratic sampling. The use of these methods did increase the number of solutions found but also increased the CPU-time and the number of leveling constraints posted. And, in all cases, ESTA^C continued to outperform EBA across all performance criteria. In analyzing the robustness of generated solutions, we found that the basic EBA procedure in fact produced solutions with greater fluidity when it was able to find a solution. However, as MCS sampling was incorporated and the number of problems solved increased, the fluidity of generated solutions simultaneously degraded, below the measured fluidity of schedules produced by ESTA^C. With regard to disruptibility, ESTA^C schedules dominated in all cases. Overall, ESTA^C was found to be a much more effective procedure.

Different aspects of the resource envelope approach and the earliest start time approach warrant further investigation. The former would benefit considerably from a more efficient envelope computation, considering that it is called into play extremely often. In the case of ESTA^C the algorithm for translating a fixed time solution into a flexible solution might be improved through use of more extended, local search techniques.

Acknowledgements

Stephen F. Smith's work is supported in part by the Department of Defense Advanced Research Projects Agency (DARPA) and the US Air Force Research Laboratory under contracts F30602-00-2-0503 and F30602-02-2-0149, and by the CMU Robotics Institute. Amedeo Cesta, Angelo Oddi and Nicola Policella's work is partially supported by ASI (Italian Space Agency) under project ARISCOM (Contract I/R/215/02). Nicola Policella is currently supported by a Scholarship from CNR on Information Technology. This work has been developed during Policella's visit at the CMU Robotics Institute as a visiting student scholar. He would like to thank the members of the Intelligent Coordination and Logistics Laboratory for support and hospitality.

References

1. 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*, 1998.
2. A. Cesta, A. Oddi, and S. F. Smith. A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristic*, 2002.
3. 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.
4. 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.
5. M.L. Ginsberg, A.J. Parkes, and A. Roy. Supermodels and Robustness. In *Proceedings of the* 15^{th} *National Conference on Artificial Intelligence, AAAI-98*, pages 334–339. AAAI Press, 1998.
6. 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.
7. 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 1643–1649, 1995.
8. H. H. El Sakkout and M. G. Wallace. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4), 2000. Special Issue on Industrial Constraint-Directed Scheduling.
9. C. Schwindt. A Branch and Bound Algorithm for the Resource-Constrained Project Duration Problem Subject to Temporal Constraints. Technical Report 544, Institut fur Wirtschaftstheorie und Operations Research, Universit at Karlsruhe, 1998.
10. S. F. Smith. OPIS: A Methodology and Architecture for Reactive Scheduling. In M. Fox and M. Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.
11. 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.