Generating Robust Schedules through Temporal Flexibility

Nicola Policella
Planning & Scheduling Team
ISTC-CNR
Rome, Italy
n.policella@istc.cnr.it

Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
Pittsburgh, USA
sfs@cs.cmu.edu

Amedeo Cesta
Planning & Scheduling Team
ISTC-CNR
Rome, Italy
a.cesta@istc.cnr.it

Angelo Oddi
Planning & Scheduling Team
ISTC-CNR
Rome, Italy
a.oddi@istc.cnr.it

Abstract

This paper considers the problem of generating *partial order schedules* ($\mathcal{POS}$), that is, schedules which retain temporal flexibility and thus provide some degree of robustness in the face of unpredictable execution circumstances. We begin by proposing a set of measures for assessing and comparing the robustness properties of alternative $\mathcal{POS}$s. Then, using a common solving framework, we develop two orthogonal procedures for constructing a $\mathcal{POS}$. 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 expands this solution to a set of resource feasible solutions in a post-processing step. We evaluate the relative effectiveness of these two procedures on a set of project scheduling benchmark problems. As might be expected, the second approach, by virtue of its more focused analysis, is found to be a more efficient $\mathcal{POS}$ generator. Somewhat counterintuitively, however, it is also found to produce $\mathcal{POS}$s that are more robust.

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.

One way to address this problem is *reactively*, through schedule repair. To keep pace with execution, the repair process must be both fast and complete. The response to a disruption must be fast because of the need to re-start execution of the schedule as soon as possible. A repair must also be complete in the sense of accounting for all changes that have occurred, while attempting to avoid the introduction of new changes. As these two goals can be conflicting, a compromise solution is often required. Different approaches exist and they tend to favor either timeliness (Smith 1994) or completeness (El Sakkout & Wallace) of the reactive response.

An alternative, *proactive* approach to managing execution in dynamic environments is to focus on building schedules that retain flexibility and are able to absorb some amount of unexpected events without rescheduling. One technique consists of factoring time and/or resource redundancy into the schedule, taking into account the types and nature of uncertainty present in the target domain (Davenport, Gefflot, & Beck 2001). An alternative technique is to construct an explicit set of contingencies (i.e., a set of complementary solutions) and use the most suitable with respect to the actual evolution of the environment (Drummond, Bresina, & Swanson 1994).

Both of these proactive techniques presume an awareness of the possible events that can occur in the operating environment, and in some cases, these knowledge requirements can present a barrier to their use. 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. We take two solution properties –the flexibility to absorb unexpected events and a solution structure that promotes localized change– as our primary solution robustness objectives, to promote both high reactivity and solution stability as execution proceeds.

We develop and analyze two methods for producing temporally flexible schedules. Both methods follow a general precedence constraint posting (PCP) strategy, which aims at the construction of a partially ordered solution, and proceeds by iteratively introducing sequencing constraints between pairs of activities that are competing for the same resources. The methods differ in the way that they detect and analyze potential resource conflicts (also referred to as resource contention peaks). The first method uses a pure least commitment approach. It computes upper and lower bounds on resource usage across all possible executions according to the exact computations recently proposed in (Muscettola 2002) (referred to as the resource envelope), and successively winnows the total set of time feasible solutions into a smaller resource-feasible set. The second method, alternatively, takes the opposite extreme approach. It utilizes a focused analysis of one possible execution (the early start time profile) as in (Cesta, Oddi, & Smith 1998; 2002), and establishes resource feasibility for a specific single-point solution (the early start time solution). This second approach is coupled with a post-processing phase which transforms this initially generated point solution into a temporally flexible schedule. These two algorithms are evaluated on a challenging benchmark from the OR literature and solution sets produced in each case compared with respect to solution robustness properties. Before describing these methods, we first define the scheduling problem of interest and propose some candidate measures for characterizing schedule robustness.

The Scheduling Problem

Given a set of activities, a set of temporal constraints and a set of resources with limited capacity, a scheduling problem consists of finding a temporal allocation of each activity such that all the resources are used consistently. In this work, we use the Resource-Constrained Project Scheduling Problem with Minimum and Maximum time lags (RPCSP/max) as a reference (Bartusch, Mohring, & Radermacher 1988). This problem involves synchronizing the use of a set of renewable resources $R = \{r_1 \ldots r_m\}$ to perform a set of activities $V = \{a_1 \ldots a_n\}$ 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 req_{ik} units of the resource r_k for all of dur_{a_j}
- a set of temporal constraints c_{ij} each defined for a pair of activities (a_i, a_j) and of the form of $c_{ij}^{min} \leq s_{a_j} - s_{a_i} \leq c_{ij}^{max}$;
- each resource r_k has an integer capacity $max_k \geq 1$;

A solution $S = (s_1, s_2, \ldots, s_n)$ to a RCPSP/max is any temporally consistent assignment of start times of all activities in V which does not violate resource capacity constraints.

CSP representation. Our work is centered on a fairly standard CSP representation of the scheduling problem. The CSP (Constraint Satisfaction Problem) representation allows us to separate the temporal constraints (a temporal constraints network) from the resource constraints.

The base of our representation is the temporal constraints network which corresponds to a Simple Temporal Problem, STP (Dechter, Meiri, & Pearl 1991). Each activity a_i to be scheduled has associated with it two relevant events: 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. We will identify the time points associated with start and end time of each activity a_i as $s_{a_i} = t_{2i-1}$ and $e_{a_i} = t_{2i}$ respectively. Additionally, two dummy time points t_0 and t_{2n+1} are used for representing the origin and the horizon of the problem, that is $t_0 \leqslant t_j \leqslant t_{2n+1} \;\; \forall j \in 1, \ldots, 2n$.

Both the duration of the activity and the constraints between any pair of activities are represented as time constraints between time points: $t_{2i} - t_{2i-1} = e_{a_i} - s_{a_i} = dur_{a_i}$ and $c_{ij}^{min} \leqslant t_i - t_j \leqslant c_{ij}^{max}$.

A directed edge-weighted graph $G_d(V_d, E_d)$ named *distance graph* is associated with the STP. In the distance graph the set of nodes V_d represents the set of time points and the set of edges E_d represents the set of constraints. In particular, for each constraint of the form $a \leq t_j - t_k \leq b$, the edge $(t_k, t_j) \in E_d$ with the label b and the edge $(t_j, t_k) \in E_d$ with the label $-a$.

According to well known properties (Dechter, Meiri, & Pearl 1991), the STP is consistent iff its distance graph G_d has no negative cycles. Let $d(t_i, t_j)$ be the length of the shortest path in G_d from the node t_i to t_j then $-d(t_j, t_i)$ and $d(t_i, t_j)$ are, respectively, the maximum and the minimum distance between the two nodes t_i and t_j, that is $-d(t_j, t_i) \leq t_j - t_i \leq d(t_i, t_j)$. In particular, when $t_j = t_0$, the interval of possible values of the time point t_i is $t_i \in [-d(t_i, t_0), d(t_0, t_i)]$. Finally, two consistent temporal scenarios (in our case a *temporal feasible solution*) are given by allocating each time point t_i to its *earliest time value*, $est(t_i) = -d(t_i, t_0)$, or to its *latest time value*, $lst(t_i) = d(t_0, t_i)$.

Superimposed on top of the temporal representation, functions $Q_j(t)$ for each resource $r_j \in R$ are used to represent resource availability over time. To model the resource usage of single activities, a value ru_{ij} is associated with any time point t_i to represent the change of resource availability. In particular, for RCPSP/max, a resource "allocation", $ru_{ij} = req_{ij}$, is associated with each activity start time and a resource "deallocation", $ru_{ij} = -req_{ij}$, is associated with the end time. Assuming S_T is the set of solutions to the STP and given a consistent assignment $\tau \in S_T$, $\tau = (\tau_1, \tau_2, \ldots, \tau_n)$, we can define the *resource profile* for any resource r_j at any time t as the sum of the values associated with the set of time points, $t_i = \tau_i$, allocated before or at the instant t, $\tau_i \leq t$:

$$Q_j^{\tau}(t) = \sum_{t_i \in \mathcal{T} \wedge \tau_i \leqslant t} ru_{ij} \tag{1}$$

This function allows us to express the resource constraint as an n-ary constraint on the set of time points $\mathcal{T}$. An assignment $\tau \in S_T$ is said to be *resource consistent* (or *resource feasible*) if and only if for each resource r_j the following property holds:

$$0 \leqslant \sum_{i | t_i \in \mathcal{T} \wedge \tau_i \leqslant t} ru_{ij} \leqslant max_j \tag{2}$$

for each solution as long as the preceding formula is verified the availability of the resource r_j will be sufficient to satisfy all the requests.

Schedule Robustness

In the realm of scheduling problems different sources of uncertainty can arise: durations may not be exactly known, resources may have lower capacity than expected (e.g., due to machine breakdowns), new tasks may need to be taken into account. We consider a solution to a scheduling problem to be *robust* if it provides an ability to absorb external events and it is structured in a way that promotes solution *stability*. In fact, the solution has to avoid amplification of the effects of a change over all its components. Keeping a solution as stable as possible has notable advantages. For instance a schedule might involve many people, each with different assigned tasks. Changing everyone's task may lead to much confusion. Our general goal is to generate schedules that achieve these solution robustness properties.

In this paper we consider the generation of temporally flexible schedules from this perspective. Within a temporally flexible schedule, each activity preserves a set of possible allocations, and these options provide a basis for responding to unexpected disruptions. More precisely, we will focus on the construction of *partially ordered solutions* (temporally consistent) that are also solutions of the overall problem (resource consistent). The aim is not to arrive to a single schedule but to instead identify a set of schedules that in the following is called *Partial Order Schedule*:

Definition 1 (Partial Order Schedule) *A Partial Order Schedule $\mathcal{POS}$ for a problem $\mathcal{P}$ is a graph, where the nodes are the activities of $\mathcal{P}$ and the edges represent temporal constraints between pairs of activities, such that any possible temporal solution is also a consistent assignment.*

Notice that the temporal constraints referred to in Definition 1 are both those defined in the problem and those added to solve it.

A $\mathcal{POS}$ provides the opportunity to reactively respond to external changes by simply propagating the effects of these changes over the Simple Temporal Problem (a polynomial time calculation), and hence can minimize the need to recompute new solutions from scratch. The challenge here is to create scheduling algorithms that create "good" $\mathcal{POS}s$, where the "goodness" of a $\mathcal{POS}$ is reflected by its *size*, the number of schedules that it "contains". In general, the larger the size of the $\mathcal{POS}$ the more flexible it is, since $\mathcal{POS}$ size is directly proportional to the ability to pick a tailored schedule for the actual evolution of the world.

The concepts introduced above are still quite vague although they give high-level intuition. We need a set of metrics for evaluating the quality of a $\mathcal{POS}$ in terms of these described features.

Evaluation Criteria. We will introduce three different metrics, the first two aim at evaluating the robustness of the solution by estimating its flexibility (size); the third estimates schedule stability.

The first measure is taken from (Aloulou & Portmann 2003) and called $flex_{seq}$. It consists of counting the *number of pairs of activities in the solution which are not reciprocally related (i.e., not ordered with respect to one another by, explicit or implicit, precedence constraints in the $\mathcal{POS}$).* This metric provides an analysis of the configuration of the solution. The rationale for this measure is that when two activities are not related it is possible to move one without moving the other one. So the higher the value of $flex_{seq}$ the lower the degree of interaction among the activities.

A second metric is taken from (Cesta, Oddi, & Smith 1998) and is based on the temporal slack associated with each activity:

$$fldt = \sum_{h \neq l} \frac{|d(e_{a_h}, s_{a_l}) + d(s_{a_l}, e_{a_h})|}{H \times n \times (n-1)} \times 100 \qquad (3)$$

where 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. An estimation of the upper bound for the horizon H is computed adding the duration of all the activities and the value of minimal distance constraints between any pair of activities. This metric characterizes 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 $fldt$, the less the risk of a "domino effect", i.e. the higher the probability of localized changes.

Whereas the previous parameters summarize the flexibility of a solution, we introduce a third measure, called *disruptibility*, to take into account the impact of disruptions on the schedule (stability):

$$dsrp = \frac{1}{n} \sum_{i=1}^{n} \frac{slack_{a_i}}{num_{changes}(a_i, \Delta_{a_i})} \qquad (4)$$

The value $slack_{a_i} = d(t_0, t_{2i}) - (-d(t_{2i}, t_0))$ represents the temporal flexibility of each activity a_i , i.e., the ability to absorb a change in the execution phase (t_{2i} is the end time of a_i and $d(t_0, t_{2i})$, $d(t_{2i}, t_0)$ are respectively its maximum and minimum possible value). 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. This function calculates the effect of propagating the value Δ_{a_i} forward, counting the number of activities which are shifted (changed) in the process. In the empirical evaluation presented later in the paper, we will assume the biggest possible shift $\Delta_{a_i} = slack_{a_i}$ when computing the number of changes. Such a metric gives an estimate of stability that incorporates the trade-off between the flexibility of each activity, $slack_{a_i}$, 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.

We now turn the attention to algorithms for generating $\mathcal{POS}$s and an analysis of how they perform with respect to these metrics.

A baseline PCP solver

To provide a basic framework for generating $\mathcal{POS}$s, we reconsider the work of some of the authors on Precedence

Constraint Posting (PCP) algorithms for solving scheduling problems (Smith & Cheng 1993; Cesta, Oddi, & Smith 1998; 2002). A PCP algorithm aims at synthesizing additional precedence constraints between pairs of activities for purposes of pruning all inconsistent allocations of resources to activities. The algorithm uses a Resource Profile (equation 1) to analyze resource usage over time and detect periods of resource conflict (contention peaks). We will see how different ways of computing and using the resource profile lead to different PCP-like algorithms.

Figure 1 shows a basic greedy algorithm for precedence constraint posting. Within this framework, 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. Given a problem, expressed as a partial ordered plan, the first step of the algorithm is to build an estimate of the required resource profile according to current temporal precedences in the network. This analysis can highlight contention peaks, where resource needs are greater then resource availability.

Conflict collection. To be more specific, we call a set of activities whose simultaneous execution exceeds the resource capacity a *contention peak*. The function Select-Conflict-Set(S_0) of Figure 1 collects all the peaks in the current schedule, ranks them, picks the more critical and select a conflict from this last peak.

The simplest way to extract a conflict from a peak is through *pairwise selection*. It consists of collecting any competing activity pairs $\langle a_i, a_j \rangle$ associated with a peak and ordering such activities with a new precedence constraint, $a_i \prec a_j$. The myopic consideration on any pair of activities in a peak can, however, lead to an over commitment. For example, consider a resource r_j with capacity $max_j = 4$ and three activities a_1, a_2 and a_3 competing for this resource. Assume that each activity requires respectively 1, 2 and 3 units of the resource. Consideration of all possible pairs of activities will lead to consideration of the pair $\langle a_1, a_2 \rangle$. But the sequencing of this pair will not resolve the conflict because the combined capacity requirement does not exceed the capacity.

An enhanced conflict selection procedure which avoids this problem is based on identification of *Minimal Critical Sets* (Laborie & Ghallab 1995) inside each contention peak. A contention peak designates a conflict of a certain size (corresponding to the number of activities in the peak). A *Minimal Critical Set*, MCS, is *a conflict such that no proper subset of activities contained in* MCS *is itself a conflict*. The idea is to represent conflicts as MCSs and eliminating them by ordering any two activities included in the MCS. In the case of the example above the only MCS is $\{a_2, a_3\}$ and both the precedence constraints $a_2 \prec a_3$ and $a_3 \prec a_2$ solve the peak.

As in previous research, we integrate MCS analysis to characterize conflicts within contention peaks. To avoid the exponential computational expense of full MCS analysis,

greedyPCP(P)
Input: A problem P
Output: A solution S
    $S_0 \leftarrow P$
    if Exists an unresolvable conflict in S_0 **then**
        return FAILURE
    else
        $C_s \leftarrow$ Select-Conflict-Set(S_0)
        if $C_s = \emptyset$ **then**
            $S \leftarrow S_0$
        else
            $(a_i \prec a_j) \leftarrow$ Select-Leveling-Constraint(S_0)
            $S_0 \leftarrow S_0 \cup \{a_i \prec a_j\}$
            $S \leftarrow$ greedyPCP(S_0)
    return S

Figure 1: Template of a greedy precedence constraint posting algorithm.

we also import two MCS *sampling procedures* from (Cesta, Oddi, & Smith 2002):

Linear sampling: instead of collecting all MCSs, we use a linear function of complexity $O(p)$, where p is the size of the peak, to sample a subset of MCSs;

Quadratic sampling: under this scheme, a larger subset of MCSs are selected using a procedure of complexity $O(p^2)$, where p is the size of the peak.

In what follow we will utilize three different operators for gathering conflicts: the simple *pairwise selection*, and the increasingly accurate *linear* and *quadratic* MCS sampling.

Conflict selection and resolution. Independent of whether conflict selection is performed directly from activity pairs or from sampled MCSs, a single conflict will be selected for resolution according to the "most constrained first" principle. Given a selected pair of conflicting activities, the order between them will be chosen according to a "least constraining" principle. The basic idea is to resolve the conflict that is the most "dangerous" and solve it with a commitment as small as possible.

More specifically, the following heuristics are assumed:

Ranking conflicts: for evaluating the contention peaks we have used the heuristic estimator K described in (Laborie & Ghallab 1995). A conflict is unsolvable if no pair of activities in the conflict can be ordered. Basically, K measures how close a given conflict is to being unsolvable.

Slack-based conflict resolution: to choose an order between the selected pair of activities we apply *dominance conditions* that analyze the reciprocal flexibility between activities (Smith & Cheng 1993). In the case where both orderings are feasible, the choice which retains the most temporal slack is taken.

It is worth underscoring that the above PCP framework establishes resource feasibility strictly by sequencing conflicting activities. It remains non-committal on activity start times. As such, PCP preserves temporal flexibility that follows from problem constraints. Further, the two heuristic choices adopt a minimal commitment strategy with respect to preserving temporal slack, and this again favors temporal flexibility.

Two Profile-Based Solution Methods

As suggested previously, we can specify dramatically different solution methods by varying the approach taken to generation and use of resource profiles. In this paper, we consider two extreme approaches: (1) a pure least commitment approach, which uses the resource envelope computation introduced in (Muscettola 2002) to anticipate all possible resource conflicts and establish ordering constraints on this basis, and (2) an "inside-out" approach which uses the focused analysis of early start time profiles that we introduced in (Cesta, Oddi, & Smith 1998) to first establish a resource-feasible early start time solution and then applies a chaining procedure to expand this early start time solution into a $\mathcal{POS}$. The subsections below consider these competing approaches in more detail.

Least-Commitment Generation Using Envelopes

The perspective of a "pure" least commitment approach to scheduling consists of carrying out a refinement search that incrementally restricts a partial solution (the possible temporal solutions $\tau \in S_T$) with resource conflicts until a set of solutions (a $\mathcal{POS}$ in our case) is identified that is resource consistent. A useful technical result has been produced recently (Muscettola 2002) that potentially can contribute to the effectiveness of this type of approach with an exact computation of the so-called Resource Envelope. According to the terminology introduced previously we can define the Resource Envelope as follows:

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

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

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

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

Integration of the envelope computation into a PCP algorithm is quite natural. It is used to restrict resource profile bounds, in accordance with the current temporal constraints in the underlying STP. In (Muscettola 2002) it is proved that it is possible to find the Resource Envelope 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. In the remainder of this section we briefly review the ideas behind computation of the resource envelope. Then we introduce some new properties we have synthesized to make the computation more efficient. Finally, we give details of how peak detection is performed starting from an envelope.

Basic envelope properties. To find the maximum (minimum) value of the resource level at any instant t most methods consider the following *set partition* of the time points (events) $\mathcal{T}$:

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

Since the events in B_t are those which happen before or at time t, they contribute to the value of the resource profile of r_j 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. 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 enumerating all the possible combinations of events in E_t. Muscettola shows that a polynomial algorithm can be found through a reduction to Max-Flow, a well-known tractable problem. The effectiveness of the reduction is due to the fact that it is possible to exploit the relations among the set of events and to consider only a subset of feasible combinations. The details of the algorithm can be found in the original paper; we simply recall here 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 $P_{max} \subseteq E_t$ ($P_{min} \subseteq E_t$), of events that gives the maximum (minimum) value of the resource level at the instant t, is computed performing a linear analysis of the residual graph. An important point is that it is not necessary to compute the resource-level envelope in all possible instants t. Indeed, you only need to compute L_j^{max} at times when either B_t or E_t changes. For any time point t_i this can only happen either at its earliest time value $est(t_i) = -d(t_i, t_0)$ or at its latest time value $lst(t_i) = d(t_0, t_i)$.

Incremental computation of resource envelopes. A potential drawback in using an envelope computation within a scheduling algorithm such as the base PCP solver is the computational burden of the Max-Flow computation. Despite being polynomial, the computational cost is significant and can become a limiting factor in the case of larger scheduling problems. In this section, we establish some properties for computing the envelope incrementally across points of discontinuity. In (Satish Kumar 2003) a method is proposed for incrementally computing the resource envelope when a new constraint is added. That method is complementary to the properties that we are proposing here. Moreover since the incremental methods to compute P_{min} and P_{max} can be obtained from each other with obvious term substitutions, we only develop the method for P_{max}.

First, we need to define the overall contribution of a time point t_i to the resource envelope value. Given a time point

t_i and a resource r_j we define its overall contribution to be the value:

$$\Sigma ru_{ij} = ru_{ij} + \sum_{\forall t_k | d(t_i, t_k) < 0} ru_{kj}$$

It is trivial to observe that a time point $t_i \in E_t$ will not belong to P_{max} if its overall contribution is negative. Indeed adding t_i at P_{max} implies a reduction of the value of the resource level at the instant t.

A first theorem allows us to restrict the set of time points for which P_{max} must be computed:

Theorem 1 *If there 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: *Reductio ad absurdum.* If $t_i \in P_{max}(E_t)$ and $t_i \notin P_{max}(E_{t+1})$ holds, then in $t+1$ the contribution of the time point t_i is negative. In turn, this entails that there must exist a time point t_k, with $ru_{kj} < -\Sigma ru_{ij}$, such that $t_k \in E_{t+1} \cap A_t$ and $d(t_i, t_k) < 0$. But the last two formulas are mutually inconsistent, thus if $d(t_i, t_k) < 0$ then $t_k \in B_t \cup E_t$. This contrasts with $t_k \in A_t$. □

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

Corollary 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 those 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 2 *An element $t \notin P_{max}(E_t)$ belongs to $P_{max}(E_{t+1})$ only if one of the following two conditions hold:*

1. *$\exists t_i^+$, with $ru_{ij} > 0$, s.t. $t_i^+ \in A_t \cap (E_{t+1} \cup B_{t+1})$*
2. *$\exists t_i^-$, with $ru_{ij} < 0$, s.t. $t_i^- \in (E_t \backslash P_{max}(E_t)) \cap B_{t+1}$.*

Proof: We prove the two cases separately:
Case 1: if $t_i \in A_t \cap (E_{t+1} \cup B_{t+1})$ then a further element is added to P_{max} only if $ru_{ij} > 0$. Indeed if $ru_{ij} \leqslant 0$ then there exists at least one production t_k^+ that is implied by t_i^-. Thus it is 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 $t_i \in (E_t \backslash P_{max}(E_t)) \cap B_{t+1}$ then a further element is added to P_{max} only if $ru_{ij} < 0$. Indeed if $ru_{ij} > 0$ then it exists 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 $t_z \in (E_t \backslash P_{max}(E_t)) \cap B_{t+1}$ s.t. $ru_{zj} < 0$. □

The above theorems allow a reduction in the computational cost of solving a given problem instance with a variant of a PCP-like solver that incorporates resource envelopes for guidance, reducing the number of times that it is necessary to recompute the set P_{max} (Theorem 1), and the size of set from which to extract it, from E_{t+1} to $E_{t+1} \backslash P_{max}(E_t)$ (Theorem 2).

Detecting peaks on resource envelopes. 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 first method (Policella *et al.* 2003) for collecting peaks consists of the following steps: (1) compute the resource envelope profile, (2) detect intervals of overallocation, and (3) collect the set of activities which can be potentially executed in such an interval. Unfortunately this approach can pick activities which are already ordered. For example, consider a problem with a binary resource and three activities a_1, a_2 and a_3 with the same interval of allocation and the precedence $a_1 \prec a_2$. In such a case the above method would collect the peak $\{a_1, a_2, a_3\}$; meanwhile, only two peaks, $\{a_1, a_3\}$ and $\{a_2, a_3\}$, should be collected in this case.

A more careful method should avoid such an aliasing effect. In particular a better method derives from considering the set P_{max}. This method is based on the particular assumption that each activity simply uses resources; 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 of every activity a_i such that the time point associated with its start time is in P_{max} but the time point associated with its end time is not, that is:

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

To avoid collection of redundant contention peaks, the extraction of the contention peak will be performed only if there exists at least one end time of an activity a_i, t_{2i}, such that it moves from A_{t-1} to $B_t \cup E_t$ and at least one start time of an activity a_j, t_{2j-1}, that moved in P_{max} since the last time a conflict peak has been collected.

Inside-Out Generation Using Early Start Profiles

A quite different analysis of resource profiles has been proposed in (Cesta, Oddi, & Smith 1998). In that paper an algorithm called ESTA (for Earliest Start Time Algorithm) was first proposed which reasons with the earliest start time profile:

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

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

This method computes the resource profile according to one precise temporal solution: the Earliest Start Time Solution. 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.

It is worth noting that the key difference between the earliest start time approach with respect to the resource envelope approach is that while the latter gives a measure of the worst-case hypothesis, the former identifies "actual" conflicts in

Chaining(P, S)
Input: A problem P and one of its fixed-times schedule S
Output: A partial order solution POS
$POS \leftarrow P$
Initialize all queues empty
for all activity a_i in increasing order w.r.t. S **do**
  for all resource r_j **do**
    $k \leftarrow 1$
    for 1 **to** ru_{ij} **do**
      $a \leftarrow LastElement(queue_{jk})$
      while $s_{a_i} \geq e_a$ **do**
        $k \leftarrow k + 1$
        $a \leftarrow LastElement(queue_{jk})$
      $POS = POS \cup \{a \prec a_i\}$
      Enqueue$(queue_{jk}, a_i)$
      $k \leftarrow k + 1$
return POS

Figure 2: Chaining algorithm.

a particular situation (earliest start time solution). In other words, the first approach says what *may* 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 limitation of this approach with respect to our current purpose is that it ensures resource-consistency of only one solution of the problem, the earliest start time solution. Using a PCP computation for solving, we always have a set of temporally consistent solutions S_T. However, ESTA will not synthesize a set of solutions for the problem (i.e., $S_T \nsubseteq \mathcal{S}$), but the single solution in the earliest start time of the resulting STP. Below, we describe a method for overcoming this limitation and generalizing an early start time solution into a partial ordered schedule ($\mathcal{POS}$). This will enable direct comparison with the $\mathcal{POS}$ produced by the envelope-based approach.

Producing a POS with Chaining. A first method for producing flexible solutions from an early start time solution has been introduced in (Cesta, Oddi, & Smith 1998). It consists of a flexible solution where a *chain* of activities is associated with each unit of each resource.

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

The first step is to consider a resource r_j with capacity max_j as a set R_j of max_j single capacity sub-resources. 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

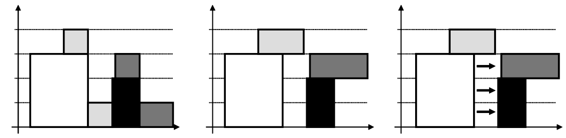

Figure 3: Chaining method: intuition.

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 are re-drawn in the way such that they are 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. This explains why the second step is needed. Indeed if the chain is built taking into account only the resource profile, there can be a problem with the relation between the light gray activity and the white one. In fact, using the chain building procedure just described, one should add a constraint between them, but that will not be sound. The second step allows this problem to be avoided, taking into account the different allocation on the set of sub-resources R_j.

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

Summary of PCP Algorithm Variants

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

	$\Delta flex_{seq}$			$\Delta fldt$			**Δdsrp**		
	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 linear	83.36	84.94	81.57	35.11	39.57	41.41	44.37	41.49	42.78
EBA+MCS quadratic	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 linear	79.79	80.41	74.97	32.42	34.87	40.94	34.75	26.74	28.39
ESTA^C+MCS quadratic	79.94	80.79	75.26	32.46	35.56	39.61	35.16	28.37	27.55

Table 1: $\Delta\mu(\mathcal{P}, \mathcal{S})$ for the three metrics $flex_{seq}$, $fldt$ and $dsrp$.

	%solved			makespan			CPU-time (secs)			posted constraints		
	J10	J20	J30	J10	J20	J30	J10	J20	J30	J10	J20	J30
EBA	77.04	50.74	43.33	58.31	96.48	118.17	0.32	3.88	24.77	11.54	33.40	63.29
EBA+MCS linear	85.19	71.11	68.89	55.29	92.65	112.14	0.77	11.35	48.89	11.12	32.87	56.84
EBA+MCS quadratic	97.78	89.63	82.22	55.47	94.03	116.10	0.91	13.21	68.22	12.38	34.98	59.64
ESTA^C	96.30	95.56	96.30	47.35	72.90	79.21	0.32	1.75	5.40	6.40	18.69	35.10
ESTA^C+MCS linear	98.15	96.67	96.67	46.63	72.45	78.45	0.34	2.14	8.08	6.23	17.49	34.07
ESTA^C+MCS quadratic	98.15	96.67	97.04	46.70	72.75	78.55	0.34	2.27	9.51	6.26	17.40	34.00

Table 2: Comparison of both the EBA and the ESTA approaches.

testing the six approaches on a significant scheduling problem benchmark: RPCSP/max.

Experimental Evaluation

This section compares the proposed set of algorithms with respect to our definition of robustness and analyzes to what extent temporally flexible solutions are also *robust* solutions able to *absorb* unexpected modifications. We compare the performance of each algorithm[1] on the benchmark problems defined in (Kolisch, Schwindt, & Sprecher 1998). This benchmark consists of three sets $J10$, $J20$ and $J30$ of 270 of problem instances of different size 10×5, 20×5 and 30×5 (number of activities $\times$ number of resources).

In a previous section we have introduced two metrics for robustness: $fldt$ and $flex_{seq}$. Both these parameters are correlated with the number of feasible solutions *contained* in a $\mathcal{POS}$. In particular, $flex_{seq}$ is directly correlated to the number of unrelated pairs of activities (no precedence constraint) in a partial order schedule. On the contrary, the disruptibility $dsrp$ is correlated with the *stability* of a solution, such that we consider executions where only one unexpected event at a time can occur (e.g., activity duration lasts longer than expected or the start time of an activity is shifted forward). We report as a result a value correlated to the average number of activities affected (number of start time changes) by the set of unexpected events.

In order to produce an evaluation of the three parameters $fldt$, $flex_{seq}$ and $dsrp$ that is independent from the problem dimension, we evaluate the following incremental parameter for each generic metric μ (i.e., $flex_{seq}$, $fldt$ or $dsrp$):

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

where $\mu(\mathcal{P})$ and $\mu(\mathcal{S})$ are respectively the values of the parameter μ for the problem $\mathcal{P}$ (the initial partial order) and its solution $\mathcal{S}$ (the final partial order). We observe that the value $\Delta\mu$ is always positive or zero. In fact, for each metric the addition of precedence constraints between activities that are necessary to establish a resource-consistent solution can only reduce the initial value $\mu(\mathcal{P})$.

The results obtained, subdivided according to benchmark set, are given in Tables 1 and 2. First, we observe that all six tested strategies are not able to solve all the problems in the benchmark sets $J10$, $J20$ and $J30$. The first column of Table 2 shows the percentage of solved problems by each strategy. This observation is particularly important, because the rest of the experimental results in this section are computed with respect to the subset of problem instances solved by all the six approaches.

Table 1 presents the main results of the paper for the six different approaches, according to the three incremental parameters $\Delta\mu$ introduced above. In each case, the lower the values, the better the quality of the corresponding solutions. In addition, Table 2 complements our experimental analysis with four more results: (1) percentage of problems solved for each benchmark set, (2) average CPU-time in seconds spent to solve instances of the problem, (3) average minimum makespan and (4) the number of leveling constraints posted to solve a problem.

From Table 1 we first observe that the ESTA^C approaches dominate the EBA approaches across all problem sets for the two metrics directly correlated to solution robustness. And this observation is confirmed in the third column ($\Delta dsrp$)

[1] All the algorithms presented in the 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.

	$J10$			$J20$			$J30$		
	scratch	incremental	$\Delta\%$	scratch	incremental	$\Delta\%$	scratch	incremental	$\Delta\%$
EBA	0.616	0.32	48.1	10.36	3.88	62.5	50.58	24.77	51.0
EBA+MCS linear	1.742	0.77	55.8	28.83	11.35	60.6	128.9	48.89	62.1
EBA+MCS quadratic	1.947	0.91	53.2	34.05	13.21	61.2	190.1	68.22	64.1

Table 3: Comparison between the CPU-time (secs) required by the EBA approaches using both the incremental and no-incremental method for computing the resource envelope.

where better values of flexibility correspond to better values of disruptibility (stability). Hence, the solutions created with ESTA^C are more appropriate to absorb unexpected events.

This fact induces further observations about the basic strategies behind the two algorithms. EBA removes all possible resource conflicts from a problem $\mathcal{P}$ by posting precedence constraints and relying on an envelope computation that produces the *tightest* possible resource-level bounds for a flexible schedule. When these bounds are less than or equal to the resource capacities, we have a resource-feasible partial order ready to *face* with uncertainty. However, in order to remove all possible conflicts EBA has to impose more precedence constraints than does ESTA^C (see column labeled with *posted constraints* in Table 2), with the risk of overcommitment in the final solution. In fact, in comparing EBA with ESTA^C, it can be seen that the EBA approach is actually less effective. It solves significantly fewer problems than ESTA^C in each problem set, obtains solutions with higher makespans, incurs higher CPU times and posts more precedence constraints. By adding MCS analysis to the EBA search configuration, we obtain a noticeable improvement of the results. In fact, in the case of quadratic sampling the number of problem solved is closer to that achieved with the ESTA^C approach. However, we pay an higher CPU time price and there are no significant improvements in the makespan and in the number of constraints posted (Table 2).

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

A note on envelope efficiency. We end the section with a final remark about our research goals. The main aim of this work has not been to find a way to beat an envelope-based algorithm, but rather to try to understand ways to use it for finding robust solutions. In this respect, EBA is the first scheduling algorithm to integrate the recent research results on exact bound computation into a scheduling framework, and, in addition, we have improved the efficiency of the envelope computation considerably with respect to our preliminary experiments (Policella *et al.* 2003). One specific result of this paper is a set of properties to reduce 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 $O(n4)$ which can be reduced to $O(n^{2.5})$ in practical cases. These computational requirements at present limit the effective application of the resource envelope. In the current implementation we use a Max-Flow method based on the *pre-flow* concept (Goldberg & Tarjan 1988). The use of the incremental properties described in a previous section speeds up the solving process by avoiding re-computation of the envelope at each step of the search. Moreover Theorem 2 allows us to apply the Max-Flow algorithm to a subset of E_{t+1}: $E_{t+1} \backslash P_{max}(E_t)$.

Table 3 reports the overall speedup obtained in solving instances of the three benchmark sets with respect to the properties expressed by Theorems 1 and 2. In particular, for each configuration of the EBA algorithm and each benchmark set ($J10$, $J20$ and $J30$) there are three different results: the average CPU-time in seconds for solving a benchmark set without incremental computation (column *scratch*), as the previous one but with the use of the incremental properties (column *incremental*) and the obtained percentage improvement over the no-incremental version of EBA (column $\Delta\%$). The results confirm the effectiveness of the incremental computation which is able to improve the CPU-time from a minimun of 48.1% to a maximum of 64.1% over the scratch computation.

Conclusion

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

To evaluate the quality of respective solutions we introduced three measures that capture desirable properties of *robust* solutions. The first two metrics ($fldt$ and $flex_{seq}$) are correlated to the degree of schedule robustness that is retained in generated solutions. The third, disruptibility $dsrp$, can alternatively be seen as the result of a simulation of solution *execution*, where we consider executions in which only one unexpected event can occur at a time. In addition, we focus our attention only to temporal disruptions: situations where an activity duration lasts longer than expected, or the start time of an activity is shifted forward.

Considering comparative performance on a set of benchmark project scheduling problems, we have shown that the two step ESTA^C procedure, which first computes a single-point solution and then translates it into a temporally flexible partial order schedule, is a more effective approach than the pure, least-commitment EBA approach. In fact, the first step preserves the effectiveness of the ESTA approach (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 produces a final schedule with better robustness properties.

Acknowledgments

Stephen F. Smith's work is supported in part by the Department of Defense Advanced Research Projects Agency and the U.S. Air Force Research Laboratory - Rome, under contracts F30602-97-2-0666 and F30602-00-2-0503, by the National Science Foundation under contract # 9900298 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 also supported by a scholarship from CNR.

Part of 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

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

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

Cesta, A.; Oddi, A.; and Smith, S. F. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the 4^{th} International Conference on Artificial Intelligence Planning Systems AIPS-98*, 241–223. AAAI Press.

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

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*.

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

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*, 1098–1104. AAAI Press.

El Sakkout, H., and Wallace, M. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*.

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, R.; Schwindt, C.; and Sprecher, A. 1998. Benchmark Instances for Project Scheduling Problems. In Weglarz, J., ed., *Project Scheduling - Recent Models, Algorithms and Applications*. Boston: Kluwer. 197–212.

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*, 1643–1649.

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*, 139–154. Springer.

Policella, N.; Smith, S. F.; Cesta, A.; and Oddi, A. 2003. Steps toward Computing Flexible Schedules. In *Proceedings of Online-2003: Workshop on "Online Constraint Solving: Handling Change and Uncertainty", Kinsale, Co. Cork, Ireland*.

Satish Kumar, T. K. 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*, 664–678. Springer.

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*, 139–144. AAAI Press.

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