J Sched
DOI 10.1007/s10951-008-0091-7

Solve-and-robustify

Synthesizing partial order schedules by chaining

Nicola Policella · Amedeo Cesta · Angelo Oddi · Stephen F. Smith

Received: 15 May 2007 / Accepted: 19 September 2008

Abstract Goal separation is often a fruitful approach when solving complex problems. It provides a way to focus on relevant aspects in a stepwise fashion and hence bound the problem solving scope along a specific direction at any point. This work applies goal separation to the problem of synthesizing robust schedules. The problem is addressed by separating the phase of problem *solution*, which may pursue a standard optimization criterion (e.g., minimal makespan), from a subsequent phase of solution *robustification* in which a more flexible set of solutions is obtained and compactly represented through a temporal graph, called a Partial Order Schedule ($\mathcal{POS}$). The key advantage of a $\mathcal{POS}$ is that it provides the capability to promptly respond to temporal changes (e.g., activity duration changes or activity start-time delays) and to hedge against further changes (e.g., new activities to perform or unexpected variations in resource capacity).

On the one hand, the paper focuses on specific heuristic algorithms for synthesis of $\mathcal{POS}$s, starting from a pre-existing schedule (hence the name *Solve-and-Robustify*). Different extensions of a technique called *chaining*, which progressively introduces temporal flexibility into the representation of the solution, are introduced and evaluated. These extensions follow from the fact that in multi-capacitated resource settings more than one $\mathcal{POS}$ can be derived from a specific fixed-times solution via chaining, and carry out a search for the most robust alternative. On the other hand, an additional analysis is performed to investigate the performance gain possible by further broadening the search process to consider multiple initial seed solutions.

A detailed experimental analysis using state-of-the-art RCPSP/max benchmarks is carried out to demonstrate the performance advantage of these more sophisticated solve and robustify procedures, corroborating prior results obtained on smaller problems and also indicating how this leverage increases as problem size is increased.

Keywords Iterative improvement techniques · Scheduling under uncertainty · Constraint-based scheduling

N. Policella (✉)
European Space Agency, 64293 Darmstadt, Germany
e-mail: nicola.policella@esa.int

A. Cesta · A. Oddi
ISTC-CNR, National Research Council of Italy, 00185 Rome, Italy

A. Cesta
e-mail: amedeo.cesta@istc.cnr.it

A. Oddi
e-mail: angelo.oddi@istc.cnr.it

S.F. Smith
The Robotics Institute, Carnegie Mellon University, 15213, Pittsburgh, PA, USA
e-mail: sfs@cs.cmu.edu

1 Introduction

The usefulness of schedules in most practical domains is limited by their brittleness. Though a schedule offers the potential for a more optimized execution than would otherwise be obtained, it must, in fact, be executed as planned to achieve this potential. In practice, this is generally made difficult by a dynamic execution environment where unexpected events quickly invalidate the predictive assumptions of the schedule and bring into question the continuing validity of the schedule's prescribed actions.

Part of the brittleness of schedules stems from reliance on a fixed-times formulation of the scheduling problem, which

designates the start and the end times of activities as decision variables and requires specific assignments to verify resource feasibility. By adopting a graph formulation of the scheduling problem instead, it is possible to produce schedules that retain temporal flexibility where problem constraints allow. In essence, such a "flexible schedule" encapsulates a set of possible fixed-times schedules, and hence is equipped to accommodate some amount of executional uncertainty. These are the reasons behind the introduction of a Partial Order Schedule or $\mathcal{POS}$ (Policella et al. 2004b).

In Policella et al. (2007), we analyzed two alternative constraint-based methodologies for synthesizing $\mathcal{POS}$s. The first method uses a pure least commitment approach where the set of all possible time-feasible schedules is successively winnowed into a smaller resource feasible set. The second method instead utilizes a focused analysis of one possible solution, and first generates a single, resource-feasible, fixed-time schedule. This point solution is then transformed into a $\mathcal{POS}$ in a second postprocessing phase referred to as *chaining*. Somewhat surprisingly, this second method was found to outperform the first one in generating robust solutions.

Following this result, this paper defines and evaluates different refinements of this two-step approach, which we refer to as *Solve-and-Robustify*. Generally speaking, *Solve-and-Robustify* is based on separating the goal of computing a solution that optimizes a classical performance objective—like minimizing makespan—from that of obtaining a robust (or flexible) solution that hedges against uncertainty. Given this separation, one important advantage of this approach is that it is possible to enlist the use of any *state-of-the-art* schedule optimization procedure in the first step. Then, in the second step, a flexible solution can be generated while attempting to preserve the optimality of the seed schedule.

In the case of binary resource problems (for instance, the job shop scheduling problem), a given fixed-times schedule establishes a unique "linearization" of all activities, and hence there is a single corresponding $\mathcal{POS}$. However, in the case of capacitated (cumulative) resource problems, the situation is different. In this setting, there are generally several distinct $\mathcal{POS}$s derivable from the same fixed-times schedule. This aspect of capacitated resource problems points up a potential limitation of the greedy, one-pass chaining procedure incorporated in Policella et al. (2007), and suggests the potential of extended chaining procedures that explore the space of possible $\mathcal{POS}$s obtainable from a given fixed-time schedule for the $\mathcal{POS}$ with the best robustness characteristics.

In this paper, we define and evaluate different iterative search extensions to the basic chaining procedure of Policella et al. (2007). Building on an initial study of such procedures in Policella et al. (2004a), we conduct an extensive experimental analysis of the relative performance of these more sophisticated chaining procedures on the state-of-the-art RCPSP/max benchmark problems. First, we demonstrate the ability to generate more robust, makespan-preserving solutions than is possible with the basic chaining, and moreover show that this performance advantage increases with larger problem sizes. We then turn attention to investigating the possibility for further improving the performance of the *Solve-and-Robustify* approach by iterating the search of possible $\mathcal{POS}$s over multiple initial seed solutions.

The remainder of the paper is organized as follows. In the next two sections, we first review the reference scheduling problem of interest and the concept of partial order schedules. Section 3 starts by introducing the basic chaining procedure and then in Sect. 4 we proceed to describe some alternatives approaches for the solving phase together with a set of, heuristically-biased, chaining search procedures. The empirical evaluation is then described in Sect. 5. Before concluding, we describe how this approach relates to the current state-of-the-art.

2 The reference problem: RCPSP/max

In this section, we recall our reference scheduling problem, the RCPSP/max. This is a highly complex scheduling problem; in fact, not only the optimization version but also the feasibility problem is NP-hard.[1] Readers interested in further details can refer to Bartusch et al. (1988).

The basic entities of this scheduling problem are *activities*, denoted as the set $V = \{a_1, a_2, \ldots, a_n\}$. Each activity a_i has a fixed *processing time*, or *duration*, d_i. Also, in the case of RCPSP/max, each activity must be scheduled without preemption.

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

$$e_i = s_i + d_i. \tag{1}$$

Feasible schedules should satisfy two types of constraints: *temporal constraints* and *resource constraints*. In their most general form, temporal constraints designate arbitrary minimum and maximum time-lags between the start times of any two activities,

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

[1] The reason for this NP-hardness result lies in the presence of maximum time-lags. In fact, maximum time lags imply the presence of deadline constraints, transforming feasibility problems for precedence-constrained scheduling into scheduling problems with time windows.

where $l_{ij}^{\min}$ and $l_{ij}^{\max}$ are the minimum and the maximum time-lag of activity a_j relative to a_i. A schedule $S = (s_1, s_2, \ldots, s_n)$ is *time feasible* if all inequalities given by the activity precedences/time-lags (2) and durations (1) hold for start times s_i.

During their processing, activities require specific resource units from a set $R = \{r_1, \ldots, r_m\}$ of resources. Resources are *reusable*, i.e., they are released when no longer required by an activity and are then available for use by another activity. Each activity a_i requires the use of req_{ik} units of the resource r_k during its processing time d_i. Each resource r_k has a limited capacity of c_k units.

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

$$\sum_{s_i \le t < e_i} req_{ik} \le c_k. \tag{3}$$

Therefore, a schedule S is called *feasible* if it is both time and resource feasible.

3 Partial order schedules

Current research approaches are based on different interpretations of robustness, e.g., the ability to preserve solution quality and/or solution stability. In this work, we pursue an execution-oriented robustness concept; a solution to a scheduling problem will be considered robust if it provides two general features: (1) the ability to absorb exogenous and/or unforeseen events without loss of consistency, and (2) the ability to keep pace with execution, guaranteeing a prompt answer to the various events.

We base our approach on the generation of flexible schedules, i.e., schedules that retain temporal flexibility. We expect a flexible schedule to be easy to change, and the intuition is that the degree of flexibility in this schedule is indicative of its robustness. Our approach adopts a graph formulation of the scheduling problem and focuses on generation of Partial Order Schedules, or $\mathcal{POS}$s. To introduce this concept we recall first the activity-on-the-node representation: given a scheduling problem P this can be represented by a graph $G_P(V_P, E_P)$, where the set of nodes $V_P = V \cup \{a_0, a_{n+1}\}$ consists of the set of activities specified in P plus two dummy activities representing the origin (a_0) and the horizon (a_{n+1}) of the schedule, and the set of edges E_P contains P's temporal constraints between pairs of activities. A solution of the scheduling problem can then be represented as an extension of G_P, where a set E_R of simple precedence constraints, $a_i \prec a_j$, is added to remove all possible resource conflicts.

As a time feasible schedule is a schedule that satisfies all the constraints defined in (1) and (2), and a feasible schedule is a schedule that is both time and resource feasible, we can define a *Partial Order Schedule* as follows:

Definition 1 (Partial Order Schedule) Given a scheduling problem P and its associated graph $G_P(V_P, E_P)$, a *Partial Order Schedule*, $\mathcal{POS}$, is a set of solutions that can be represented by a graph $G_{POS}(V_P, E_P \cup E_R)$ such that any *time feasible* schedule defined by the graph is also a *feasible* schedule.

A $\mathcal{POS}$ is then a set of activities partially ordered such that any possible complete order that is consistent with the initial partial order, is a resource and time feasible schedule. In the following, two concepts connected to the partial order schedule are introduced: the earliest start schedule and the makespan of a $\mathcal{POS}$.

Definition 2 The earliest start schedule of a $\mathcal{POS}$, $ES(\mathcal{POS})$, is defined as the schedule $S = (s_1, s_2, \ldots, s_n)$ in which each activity is scheduled to start at its earliest start time, that is, $s_i = est(a_i)$ for $1 \le i \le n$.

Definition 3 The makespan of a partial order schedule is defined as the makespan of its earliest start schedule, that is,

$$mk(\mathcal{POS}) = mk\big(ES(\mathcal{POS})\big) = \max_{a_i \in V}\big\{est(a_i) + d_i\big\}.$$

To evaluate different $\mathcal{POS}$s we refer to the concept of robustness. Intuitively, the robustness of a partial order schedule is tightly related to the set of solutions that it represents. One key feature is the number of solutions that a $\mathcal{POS}$ contains; the greater the number of solutions that it contains, the greater the expected ability of the $\mathcal{POS}$ to absorb uncertainty in the temporal constraints of the problem. A second important characteristic of the solutions clustered into a partial order schedule is the distribution of these alternative solutions. This distribution will be the result of the configuration given by the constraints present in the solution. To take into account these two aspects of robustness, we introduce two $\mathcal{POS}$ quality metrics in Sect. 5.1.

Given a possible *temporal change* (e.g., an activity delay or an extension of an activity duration) a new schedule can be obtained by just propagating this change through the $\mathcal{POS}$. Therefore, a partial order schedule provides an efficient means for finding a new solution. To guarantee a *prompt answer* to possible changes it is necessary to evaluate the cost of propagating them through the $\mathcal{POS}$. To fully propagate the effects of the new edge through the graph, it is necessary to achieve *arc-consistency* of the graph (i.e., to ensure that any activity has a legal allocation with respect to the temporal constraints of the problem). The Bellman–Ford Single-Source Shortest Path algorithm which finds the shortest paths among all vertexes in a weighted, directed

graph, achieves this condition with a *polynomial* time complexity, $O(nm)$, where n is the number of activities and m is the number of constraints. Better results can be obtained by considering incremental algorithms that avoid computation of the solution from scratch at each step. For instance, Cesta and Oddi (2001) introduce an incremental algorithm with time complexity $O(\delta_n \delta_m)$, where $\delta_n \leq n$ is the number of vertices (activities) whose shortest path has changed, and $\delta_m \leq m$ is the number of arcs (constraints) with at least one affected end-point.[2]

In the remainder of the section we briefly recall the basic chaining method used to synthesize $\mathcal{POS}$s in Policella et al. (2004b, 2007), and analyze some particular features of the $\mathcal{POS}$s obtained via chaining. These features are preparatory for the solving schema discussed in Sect. 4.

3.1 The chaining algorithm

We describe here the basic chaining method presented in Policella et al. (2004b). This method itself is a generalization of one first introduced by Cesta et al. (1998). Given a solution, a transformation method, named *chaining*, is defined that proceeds to create sets of chains of activities. Algorithm 1 describes the chaining process; it uses a set of chains 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. When an activity is allocated to a chain, a new constraint between this activity and the previous one in the chain is posted. Let m and $\max_{\text{cap}}$ be, respectively, the number of resources and the maximum capacity among the resources. Then the complexity of the chaining algorithm is $O(n \log n + n \cdot m \cdot \max_{\text{cap}})$. In the following section, we examine the concept of *Chaining Form* solutions from the broader perspective of generating robust $\mathcal{POS}$s.

Algorithm 1 Chaining procedure

Require: a problem P and one of its fixed-times schedules S
Ensure: A partial order solution POS
  $POS \leftarrow P$
  Sort all the activities according to their start times in S
  Initialize the all chains empty
  for all resource r_j **do**
    for all activity a_i **do**
      for 1 **to** req_{ij} **do**
        $k \leftarrow SelectChain(a_i, r_j)$
        $a_k \leftarrow last(k)$
        $AddConstraint(POS, a_k \prec a_i)$
        $last(k) \leftarrow a_i$
  return POS

[2]This algorithm cannot be extended to the case where a constraint is removed from the graph. This can be the case, for instance, due to reduction of an activity duration. This problem can be overcome by employing two different approaches to managing the solution during execution: either by recomputing a new solution from scratch by using a complete algorithm or simply by doing nothing.

3.2 Partial order schedules in chaining form

The concept of chaining form refers to a $\mathcal{POS}$ in which a *chain* of activities is associated with each unit of each resource or, in other words, a $\mathcal{POS}$ which can be obtained by applying a chaining method to a fixed-time schedule. More formally we have:

Definition 4 (Partial Order Schedule in Chaining Form) Given a scheduling problem P and the associated $\mathcal{POS}$, this is in *Chaining Form* if for each unit j of a resource r_k it is possible to identify a set (possibly empty) of activities $\{a_{j,0}, a_{j,1}, \ldots, a_{j,N_j}\}$ such that (1) $a_{j,i-1}$ will be executed before $a_{j,i}$, $a_{j,i-1} \prec a_{j,i}$ for $i = 1, \ldots, N_j$ and (2) for each resource r_k, the activity $a_{i,j}$ is allocated to $req_{a_{i,j},k}$ different chains.

The second condition states that each activity will be allocated to as many chains as necessary to fulfill its resource requirements. Figure 1 shows a partial order schedule in chaining form for a problem with a single resource r_k with capacity $c_k = 4$. The bold (horizontal) arcs represent the set of chains and the thin arcs designate the constraints defined in the problem (that is, $a \prec b$ and $c \prec e$). The size of each activity reflects both its duration and its resource requirement, respectively, the length represents the duration, and the height represents the number of units requested. Hence, the gray activities require more than one unit of resource. This implies that both of them will be allocated to more than one chain.

Now we show that any given $\mathcal{POS}$ is expressible in chaining form. Of course, a solution obtained by the chaining procedure is by definition in this form. We then prove that any partial order schedule $\mathcal{POS}$ admits at least an equivalent $\mathcal{POS}$ in chaining form.[3]

Theorem 1 *Given a partial order schedule $\mathcal{POS}$ there exists a partial order schedule in chaining form, $\mathcal{POS}^{\text{ch}}$, that represents at least the same set of solutions.*

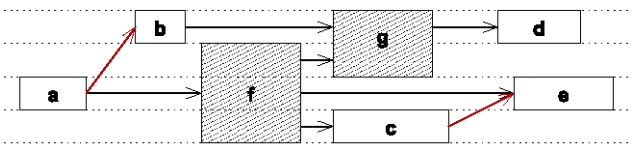


Fig. 1 A partial order schedule in chaining form

[3]An analogous result has been proved in Leus and Herroelen (2004).

Proof Let $\overline{POS}(V_P, \overline{E})$ be the transitive closure of the graph $\mathcal{POS}$, where $\overline{E} = E_P \cup E_R \cup E_T$ and E_T is the set of simple precedence constraints $a_h \prec a_l$ added to $\mathcal{POS}$ when there is a precedence constraint between a_h and a_l induced by the constraints represented in the set $E_P \cup E_R$. It is always possible to construct a graph $\mathcal{POS}^{\text{ch}}(V_P, E_P \cup E^{\text{ch}})$ with $E^{\text{ch}} \subseteq \overline{E}$ such that POS^{ch} represents at least the same set of solutions of $\mathcal{POS}$. In fact, given the set $\overline{E}$, for each resource r_k, we can always select a subset of simple precedence constraints $E_k^{\text{ch}} \subseteq \overline{E}$ such that it induces a partition of the set of activities requiring the same resource r_k into a set of chains. In particular, for each resource r_k and unit j of resource r_k, it is possible to identify a set (possibly empty) of activities $\{a_{j,0}, a_{j,1}, \ldots, a_{j,n_j}\}$ such that $(a_{j,i-1}, a_{j,i}) \in E_k^{\text{ch}} \subseteq \overline{E}$ with $i = 1, \ldots, n_j$ and $E^{\text{ch}} = \bigcup_{k=1}^{m} E_k^{\text{ch}}$.

Proof by contradiction: let us assume as not possible the construction of such a $\mathcal{POS}^{\text{ch}}$. Then, there is at least one resource r_k for which there is an activity a_k which requires the use of r_k and does not belong to any chain of r_k. This means that there exists at least a set of mutual overlapping activities $\{a_{i1}, a_{i2}, \ldots, a_{ip}\}$, where each activity a_{ij} belongs to a different chain and $p = c_k$, such that the set $\{a_k, a_{i1}, a_{i2}, \ldots, a_{ip}\}$ represents a forbidden set. This last fact contradicts the hypothesis that $\mathcal{POS}$ is a partial order schedule. Thus, it is always possible to build a $\mathcal{POS}^{\text{ch}}$ from a $\mathcal{POS}$ with $E^{\text{ch}} \subseteq \overline{E}$. □

An interesting property of applying a chaining procedure is related to the common objective of producing solutions which minimize makespan values:

Property 1 Given a fixed-times schedule S and a related $\mathcal{POS}_S$ obtained by applying a chaining method:

$$mk\big(ES(\mathcal{POS}_S)\big) \leq mk(S), \tag{4}$$

i.e., the makespan of the earliest solution of $\mathcal{POS}_S$ is not greater than the makespan of the fixed-time solution S.

Equation (4) can be explained by considering that, by definition, S is one of the solutions represented by $\mathcal{POS}_S$. Practically, since only simple precedence constraints already contained in the input solution S are added, the makespan of the output solution will not be greater than the original one. Thus, in the case of a makespan objective, the robustness of a schedule can be increased without degradation to its solution quality. Moreover, because the original solution is always included in the generated $\mathcal{POS}$, it is intuitive that in a $\mathcal{POS}$ a fixed time solution which preserves the original characteristics will always exist.

Thanks to the results shown in this section, we can restrict our attention to the design of procedures that explore the space of partial order schedules in such a form. This possibility will be exploited in Sect. 4.2.

4 Solve-and-robustify: a two step approach to $\mathcal{POS}$ generation

The Solve-and-Robustify approach consists of a two step procedure where fixed-time solutions built in the first step are used to obtain partial order schedules in the second step. This approach permits separate analysis of the two key objectives: finding a solution that optimizes performance and building a flexible schedule that anticipates the need for change. In this section, we recall the basic chaining method used to synthesize a $\mathcal{POS}$ given a fixed time solution. We then analyze more deeply the characteristics of the partial order schedules produced with this approach—$\mathcal{POS}$s in *chaining form*. Finally, in order to obtain higher quality solutions, we introduce two alternative iterative chaining algorithms.

4.1 Solving the problem

As indicated previously, the *Solve-and-Robustify* approach is based on the assumption of independence between classical scheduling objectives—like minimizing makespan—and objectives relating to the generation of a robust, flexible solution. Even though this assumption is, in general, not true, *Solve-and-Robustify* is an interesting heuristic strategy for a couple of reasons. First, any of a number of *state-of-the-art* schedulers that have been designed to obtain optimal solutions to particular performance objectives can be exploited in the first "solving" step. Then, in a next step, a flexible solution can be generated that is biased by and attempts to preserve the optimality of the starting schedule. Second, the characteristics of the $\mathcal{POS}$ produced in the second step tend to preserve optimality in circumstances where there is a reasonably low degree of environmental uncertainty. In such circumstances, deviations from expected constraints tend to result in new allocation decisions that are close to those in the original schedule and hence close in expected outcome. Broadly speaking, the closer the two allocations are to each other, the less loss that is incurred in objective function values.

In Policella et al. (2007) we demonstrated the comparative strength of the *Solve-and-Robustify* approach in generating schedules with good robustness properties over the more conventional "least commitment" approach that has been favored historically in planning and scheduling research. In these initial configurations of *Solve-and-Robustify*, several versions of a heuristic solver called ESTA (Cesta et al. 1998) were coupled with a simple chaining procedure. In brief, ESTA is a constraint based scheduling procedure which exploits the analysis of resource profiles to detect and eliminate resource conflicts by posting ordering constraints in an underlying temporal constraint network representation of the scheduling problem.

In this paper, we take the best performing variant of ESTA found in Policella et al. (2007) as our starting point for the solving step, and focus on defining and evaluating more effective variants of the *Solve-and-Robustify* approach. We focus first on improving the robustify phase, and evaluate the use of a set of more sophisticated chaining procedures. By settling on the use of ESTA as the "solver", we provide a common initial solution to all chaining alternatives for purposes of comparison (recognizing that we could substitute any relevant scheduling algorithm to potentially boost performance in any particular domain).

After assessing the performance of alternative chaining procedures, we broaden our analysis to consider more sophisticated versions of the solving phase. Because the Solve-and-Robustify approach can be seen as an open loop cascade of a "solver" and a "robustifier" module, we analyze the possibility of seeding the chaining procedures with different starting point solutions. The analysis in Sect. 5.3 considers two alternatives which use a randomized version of ESTA:[4]

1. The use of an iterated version of ESTA, called the ISES procedure (Cesta et al. 2002), in the solving phase. This algorithm is an iterative method which at each step utilizes the randomized version of ESTA to produce different solutions.
2. A new procedure, GRASP, where the whole process of solving plus robustifying is iterated in order to optimize the quality of the final $\mathcal{POS}$. In this case, at each step, once a solution is produced via the randomized version of ESTA, a $\mathcal{POS}$ is synthesized with a chaining procedure.

By using the first alternative, we create a loop around the solver and therefore we try to understand how much we can influence robustness by searching for a better starting fixed-times schedule. In contrast, the second alternative implements a typical "Greedy Randomized Adaptive Search Procedure", or GRASP, meta-heuristic (see Resende and Ribeiro 2002). It creates a loop around the basic cascade and iterates the two step approach of generating and robustifying (different) fixed-time schedules. The goal of this second analysis is to evaluate different possible component extensions of the Solve-and-Robustify schema and to weigh which of these is more important for the overall efficiency of the approach.

4.2 Increasing robustness features through iterative chaining

A chaining procedure is a post-processing step which dispatches (or allocates) tasks to specific resource units once that a resource feasible (fixed-times) solution has been built. In the basic implementation (Algorithm 1) this dispatching process is carried out in a specific deterministic manner; the $SelectChain(a_i, r_j)$ function is defined such that the next activity a_i is always dispatched to the first available resource unit (chain) associated with its required resource r_j. However, since there are generally choices concerning how to dispatch activities to resource units, it is possible to generate different $\mathcal{POS}$s from a given initial fixed-times schedule, and these different $\mathcal{POS}$s can be expected to have different robustness properties. In this section, we follow up on this observation, and define a set of procedures for searching this space of possible chaining solutions. The goal in each case is to maximize the size (i.e., number of contained solutions) of the final $\mathcal{POS}$ produced by the chaining process, since a maximally sized $\mathcal{POS}$ offers the greatest possibility to accommodate temporal uncertainty. Given the results of the previous section, we can search this space of possible $\mathcal{POS}$s with assurance that the "optimal" solution is reachable.

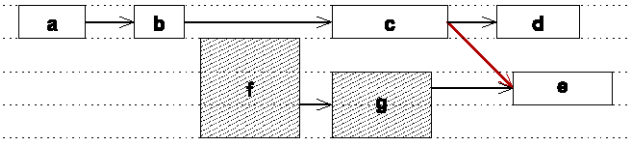
We adopt an Iterative Sampling search procedure as a basic framework for exploring the space of the possible $\mathcal{POS}$s. Specifically, the chaining procedure (Algorithm 1) is executed num_{iter} times starting from the same initial fixed-time solution, and non-determinism is added to the strategy used by $SelectChain(a_i, r_j)$ to obtain different $\mathcal{POS}$s across iterations. Each $\mathcal{POS}$ generated is evaluated with respect to some designated measure of robustness, and the best $\mathcal{POS}$ found overall is returned at the end of the search.

As a baseline for comparison, we define an initial iterative search procedure in which $SelectChain(a_i, r_j)$ allocates activities to available chains in a completely random manner. Though this completely random iterative procedure will certainly examine a large number of candidate $\mathcal{POS}$s, it does so in an undirected way and this is likely to limit overall search effectiveness.

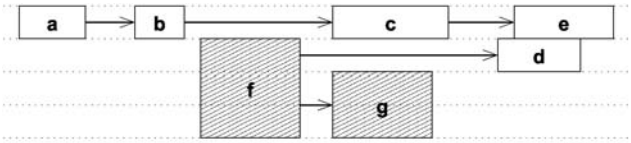
A more effective procedure can be obtained by using a heuristic to bias the way in which chains are built. To design an informed heuristic for dispatching activities to chains, it is useful to examine the structure of solutions produced by the chaining procedure. Consider the $\mathcal{POS}$ in Fig. 1. Note that both of the activities requiring multiple resource units (the gray activities) and the precedence constraints between activities that are situated in different chains tie together the execution of different chains. These interdependencies, or *synchronization points*, tend to degrade the flexibility of a solution. In fact, if we consider each single chain as being executed as a separate process, each synchronization point will mutually constrain two, otherwise independent processes. When an unforeseen event occurs and must be taken into account, the presence of these points will work against the $\mathcal{POS}$s ability to both absorb the event and retain flexibility for future changes. Hence, it is desirable to minimize the number of synchronization points where possible.

To summarize, synchronization points can originate from one of two different sources:

[4] So designed to produce different solution each time it is invoked.



(a) A more flexible $\mathcal{POS}$. The execution of the activities on the first chain, a, b, c, and d, is completely independent from the two activities f and g.



(b) An even more flexible $\mathcal{POS}$. The flexibility is increased putting the pairs activities (a, b) and (c, e) on the same chain as a precedence constraint between them is already defined in the problem.

Fig. 2 Two different partial order schedules obtained from the same fixed-time solution of the one in Fig. 1

– a constraint defined in the problem which relates pairs of activities belonging to different chains;
– an activity that requires two or more resource units and/or two or more resources will be part of two or more chains.

In the first case, the synchronization point is strictly a consequence of the problem. However, in the second case, the synchronization point could follow from the way the chains are built and might be preventable. For example, consider the $\mathcal{POS}$ given in Fig. 2(a). Here a more flexible solution than the one previously discussed in Fig. 1 is obtained by simply allocating the two gray activities to the same subset of chains. In the $\mathcal{POS}$ in Fig. 1, the two gray activities span all four chains. They effectively split the solution into two parts, and the entire execution phase will depend on the execution of these two activities. On the contrary, choosing to allocate these activities to common chains results in at least one chain that can be independently executed.

Based on this observation, we define a first heuristic chain selection procedure that favors allocation of activities to common chains (*maximizing common chains*, MAXCC). Under this procedure, allocation of an activity a_i proceeds according to the following four steps:

(1) An initial chain k is randomly selected from among those available for a_i and the constraint $a_k \prec a_i$ is posted, where a_k is the last activity in chain k.
(2) If a_i requires more than one resource unit, then the remaining set of available chains is split into two subsets: the set of chains which has a_k as last element, C_{a_k}, and the set of chains which does not, $\bar{C_{a_k}}$.
(3) To satisfy all remaining resource requirements, a_i is allocated first to chains belonging to the first subset, $k' \in C_{a_k}$ and,
(4) In case this set is not sufficient, the remaining units of a_i are then randomly allocated to the first available chains, k'', of the second subset, $k'' \in \bar{C_{a_k}}$.

To see the benefits of using this heuristic, let us reconsider once again the example in Fig. 1. As described above, the critical allocation decisions involve the two gray activities which require 3 and 2 resource units, respectively. If the first resource unit selected for the second gray activity happens to coincide with one that is already allocated to the first gray activity, then the use of the above heuristic will force selection of a second common chain for the second gray activity. A possible result of using this heuristic chain selection procedure is, in fact, the $\mathcal{POS}$ in Fig. 2(a).

The example in Fig. 2(a) also illustrates a second anomaly that can be observed in chaining form $\mathcal{POS}$s. Notice the presence of a synchronization point due to the problem constraint between activity c and e. While such problem constraints cannot be eliminated, they can, in fact, be made redundant if both activities can be allocated to the same chain(s). This observation leads to the definition of a second heuristic chain selection procedure, which augments the first by replacing the random selection of the first chain for a given activity with a more informed choice that takes into account existing ordering relations with those activities already allocated in the chaining process (*minimizing interdependencies*, MINID). Therefore, the above step (1) is replaced by the following sequence:

(1a) The chains k for which their last element, $last(k)$, is already ordered w.r.t. activity a_i, are collected in the set P_{a_i}.
(1b) If $P_{a_i} \neq \emptyset$, a chain $k \in P_{a_i}$ is randomly picked; otherwise, a chain k is randomly selected among the available ones.
(1c) A constraint $a_k \prec a_i$ is posted, where a_k is the last activity of the chain k.

At this point the procedure proceeds with the steps (2), (3), and (4) described above. Figure 2(b) shows the result of applying of this second heuristic chain selection procedure to our example. Since both activity c and activity e are dispatched to the same chain the synchronization point present in Fig. 2(a) is eliminated.

5 Experimental evaluation

To evaluate the effectiveness of the Solve-and-Robustify alternatives, we consider their performance on four sets of RCPSP/max problems: $j10$, $j20$, $j30$, and $j100$ with problems of, respectively, 10, 20, 30, and 100 activities and 5

resources (Kolisch et al. 1998). All problem sets were generated by PROGEN/MAX, a flexible random networks generator capable of creating project scheduling problems of varying structure, constrainedness and difficulty. Due to differences in the generation parameters used in each case, there are important differences in the characteristics of the problems in each set. The parameter settings used to generate $j10$, $j20$, and $j30$ are closer than $j100$ to the settings which produce the "hardest possible" problems. In particular, the problems in $j10$, $j20$, and $j30$ exhibit higher levels of resource conflict than those in $j100$, along with increased parallelism in project activities (i.e., increased sequencing flexibility). Given these properties, we can expect a larger search space in solving problems from $j10$, $j20$, and $j30$ than for problems in $j100$, since a greater number of ordering decisions are needed to build up a feasible solution.

The different procedures compared below were all implemented in C++ and run on a Pentium 4-1500 MHz with 512 MB of RAM (Linux operating system).

5.1 Metrics for comparing partial order schedules

As introduced earlier, the quality of a partial order schedule is related to the set of solutions that it represents. In particular, there are two important aspects: the size of this set and the distribution of these alternative solutions. To consider these aspects we here introduce two $\mathcal{POS}$ metrics. The first is the *flex* metric specified in Aloulou and Portmann (2005):

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

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

Unfortunately, *flex* gives only a qualitative evaluation of the quality of a $\mathcal{POS}$. In fact, it only considers whether a "path" exists between two activities, and not how flexible the path is. Even though this measure may be sufficient for a scheduling problem with no time lag constraints like the one used in Aloulou and Portmann (2005), it is insufficient for our purposes. In a problem like RCPSP/max, it is necessary to integrate *flex* with some measure that also takes into account this quantitative aspect of the problem (or solution).

Before introducing such a measure, however, it is worth noting that in order to compare two or more $\mathcal{POS}$s it is also necessary to have a finite number of solutions. This is possible if it can be assumed that all activities in a given problem must be completed within a specified, finite, horizon. *Hence, it follows that within the same horizon H, the greater the number of solutions represented in a $\mathcal{POS}$, the greater its robustness*. The goal then is to compute a fair bound (or horizon) which is large enough to allow all activities to be executed, and therefore does not introduce any bias in the evaluation of a solution. One possible value is the following:

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

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

The presence of a fixed horizon allows the use of an alternative metric introduced in Cesta et al. (1998): this is defined as the average width, relative to the temporal horizon, of the temporal slack associated with each pair of activities (a_i, a_j):

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

where H is the horizon of the problem defined above, n is the number of activities, $slack(a_i, a_j)$ is the width of the allowed distance interval between the end time of activity a_i and the start time of activity a_j, and 100 is a scaling factor.[5] This metric characterizes the *fluidity* of a solution, i.e., the ability to use flexibility to absorb temporal variation in the execution of activities and avoid deleterious domino effects. The higher the value of *fldt*, the lower the risk of cascading change and the higher the probability of localized changes.

In order to produce an evaluation that is independent from the problem dimension, we normalize the values computed for both *fldt* and *flex* with respect to a suitable upper bound. The upper bound is obtained for each metric $\mu()$ by considering the value $\mu(P)$ that is the quality of the input problem P. In fact, for each metric the addition of those precedence constraints between activities that are necessary to establish a resource feasible solution can only reduce the initial value $\mu(P)$, that is, $\mu(P) \geq \mu(S)$. Then, given the problem P, the normalized value of the metric $\mu()$ for the solution S is:

$$\left|\mu(S)\right| = \frac{\mu(S)}{\mu(P)}, \tag{8}$$

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

where the higher the normalized value (for either |*flex*| or |*fldt*|) the better.

It is worth remarking that both metrics are used to analyze the static characteristics of $\mathcal{POS}$s. Of course, these metrics also give some insight into the reactivity that a given $\mathcal{POS}$ provides (in terms of the time needed to reply to an unexpected event); the more flexible the $\mathcal{POS}$, the more likely that temporal variation can be efficiently accommodated (as seen in Sect. 3, this can be done by using a polynomial constraint propagation phase). As mentioned earlier, the time required to respond will also depend on the type of propagation strategy used to execute the $\mathcal{POS}$s. In Rasconi et al. (2008), an empirical framework has been designed to evaluate $\mathcal{POS}$s execution with respect to the reaction time and its relation with *flex* and *fldt* values.

5.2 Analyzing the use of iterative chaining methods

This section is dedicated to the analysis of iterative chaining procedures. As mentioned earlier, the algorithm used in the "solve" phase to obtain initial fixed-times solutions is the ESTA procedure of Cesta et al. (1998). Even though, ESTA is a greedy method, it is very effective as it solves, respectively, 96.3%, 95.6%, 96.3%, and 99.3% of *j*10, *j*20, *j*30, and *j*100. In Policella et al. (2007) we considered different ESTA versions based on three different heuristics. Here we just consider the best of these alternatives.[6] Please refer to Policella et al. (2007) for further details.

In particular, we compare the basic chaining procedure, *chn* (see Algorithm 1), against the following three iterative sampling chaining methods (each optimized w.r.t. either *fldt* or *flex*):

- the basic randomized versions, $rndm_{fldt}$ and $rndm_{flex}$;
- the first heuristic biased versions aimed at maximizing chain overlap between activities that require multiple resource units, MAXCC_{fldt} and MAXCC_{flex};
- the enhanced heuristic biased versions which add consideration of extant activity ordering constraints, MINID_{fldt} and MINID_{flex}.

For each procedure, five metric values are shown: the flexibility (|*flex*|), the fluidity (|*fldt*|), the CPU-time in seconds (*cpu*), the number of precedence constraints posted (*npc*), and the makespan (*mk*). With respect to the CPU time, we include both the time to find an initial fixed-times solution and the time required by the chaining procedure. In the case of iterative procedures, the values shown reflect 100 iterations.

Tables 1 to 4 present, respectively, the results obtained on the different benchmark sets. The first observation is that all search procedures outperform the basic chaining procedure: it is clearly worthwhile to explore the space of possible $\mathcal{POS}$s derivable from a given fixed-times solution. Also, all search strategies produce some amount of improvement in solution makespan as an interesting side benefit.

Table 1 *j*10: performance of the algorithms

*j*10	|*flex*|	|*fldt*|	*cpu*	*npc*	*mk*
chn	0.19	0.68	0.02	6.48	49.49
$rndm_{fldt}$	0.20	0.70	0.13	6.47	49.30
$rndm_{flex}$	0.22	0.69	0.12	6.44	49.38
MAXCC_{fldt}	0.25	0.71	0.12	6.45	49.27
MAXCC_{flex}	0.26	0.70	0.12	6.41	49.31
MINID_{fldt}	0.29	0.72	0.12	5.90	49.12
MINID_{flex}	0.29	0.71	0.12	5.92	49.19

Table 2 *j*20: performance of the algorithms

*j*20	|*flex*|	|*fldt*|	*cpu*	*npc*	*mk*
chn	0.16	0.65	0.12	18.86	83.97
$rndm_{fldt}$	0.17	0.70	0.44	18.79	83.45
$rndm_{flex}$	0.18	0.68	0.43	19.93	83.82
MAXCC_{fldt}	0.21	0.69	0.43	20.13	83.55
MAXCC_{flex}	0.22	0.68	0.43	19.83	83.78
MINID_{fldt}	0.28	0.71	0.42	16.95	83.06
MINID_{flex}	0.29	0.71	0.42	16.67	83.18

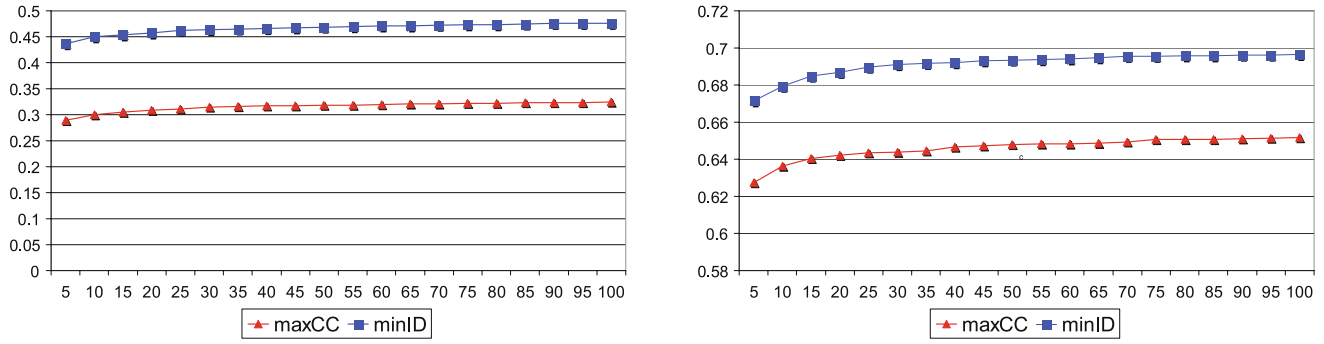
Table 3 *j*30: performance of the algorithms

*j*30	|*flex*|	|*fldt*|	*cpu*	*npc*	*mk*
chn	0.25	0.59	0.41	38.7	107.08
$rndm_{fldt}$	0.26	0.63	1.28	44.7	106.13
$rndm_{flex}$	0.28	0.62	1.28	44.8	106.78
MAXCC_{fldt}	0.31	0.65	1.27	39.6	106.42
MAXCC_{flex}	0.33	0.63	1.27	39.1	106.67
MINID_{fldt}	0.46	0.70	1.21	28.9	105.41
MINID_{flex}	0.48	0.67	1.21	28.3	105.71

Table 4 *j*100: performance of the algorithms

*j*100	|*flex*|	|*fldt*|	*cpu*	*npc*	*mk*
chn	0.05	0.50	2.07	71.97	440.79
$rndm_{fldt}$	0.05	0.53	12.38	54.75	439.91
$rndm_{flex}$	0.07	0.52	12.28	54.84	441.68
MAXCC_{fldt}	0.06	0.55	12.07	54.54	439.80
MAXCC_{flex}	0.07	0.53	12.06	54.57	440.38
MINID_{fldt}	0.22	0.65	12.07	39.21	437.10
MINID_{flex}	0.22	0.64	12.05	39.34	437.93

[6] The same variant is considered for the randomized versions of ESTA below.



(a) Flexibility vs number of iterations

(b) Fluidity vs number of iterations

Fig. 3 Iterative sampling efficiency for the benchmark $j30$

Focusing our attention on the benchmark $j30$ (Table 3), we can notice that the iterative sampling procedure with heuristic bias, MAXCC$_{flex}$, improves 30% over the basic chaining results, while the version using the enhanced heuristic, MINID$_{flex}$, obtains a gain of about 92% (from 0.25 to 0.48). The informed selection of the first chain thus is clearly a determining factor in achieving high quality solutions. This is confirmed by comparing the two heuristic versions with the simple randomized version (*rndm*$_{flex}$): the improvement is of about 18% in the case of MAXCC$_{flex}$ and of about 71% in the case of MINID$_{flex}$. The same behavior can be seen for the procedures with the *fldt* parameter. In this case, improvement ranges from about 10% for the first heuristic MAXCC$_{fldt}$, to about 16%, for MINID$_{fldt}$.

In general, the contribution of the heuristic increases according to the size of the problem. For the *flex* metric, we see that the procedure MINID$_{flex}$ gives an improvement of 53%, 81%, 92%, and 340%, respectively, for $j10$, $j20$, $j30$, and $j100$. For the *fldt* metric, we have instead an improvement of 6%, 9%, 16%, and 30%. Though the contribution is smaller in the latter case, note that for the *fldt* metric the simple chaining procedure *chn* alone is able to obtain a noticeable average value: 0.50 in the worst case (Table 4).

Another discriminating aspect of the enhanced heuristic MINID is its ability to take advantage of pre-existing precedence constraints and reduce the number of posted constraints[7] (i.e., see *npc* column in Table 4). This effect, as might have been predicted, was seen to improve both the fluidity and (especially) the flexibility values. Moreover, use of the enhanced heuristic also yielded the most significant reduction in solution makespan. Intuitively, the smaller number of posted constraints contributes to compression of the critical path. On the other hand, use of the iterative procedure incurs a non negligible additional computational cost.

Figure 3 highlights a further aspect which differentiates the heuristic biased iterative procedures from the pure randomized procedure. This picture plots the average value of the best solution found on benchmark $j30$ by each iterative procedure as the search progresses (with respect to the number of iterations). Figure 3(a) represents the results obtained when the metric *flex* is taken into account while in Fig. 3(b) the procedures aim at optimizing the *fldt* value. The heuristic based on minimizing the interdependencies is seen to find better solutions at a much faster rate than the other one. It is clear that the use of heuristic bias focuses the search on a more significant region of the search space with respect to both robustness metrics, and that this bias both accelerates and enhances generation of better solutions.

5.3 Investigating the use of different fixed times solutions

In this section we broaden our analysis of the Solve-and-Robustify schema to consider different starting point solutions. In particular, we consider two alternatives (see Sect. 4.1). In the first approach, the ISES algorithm (described in Cesta et al. 2002) is used as solving method, while in the GRASP approach we use a randomized version of ESTA. For each method we consider two different variants corresponding to the parameter (*flex* or *fldt*) that is optimized.

Table 5 shows the results for the benchmark $j30$. Considering our two robustness metrics, it is possible to notice a twofold behavior. On the one hand, the values of *flex* are close to or equal to the best value, 0.48, obtained in Table 3. This is due to the fact that both procedures make use of the iterative chaining method MINID which plays an important rule in increasing the flexibility (in terms of *flex*) of the final partial order schedule.[8] On the other hand, the same behavior is not confirmed by the results obtained when optimizing

[7] Note that in any of the chaining methods a precedence constraint $a_k \prec a_i$ is posted iff a_k and a_i are not ordered already.

[8] Of course, there is a noticeable difference in terms of CPU time, 8.65 against 1.21 seconds.

Table 5 Comparison between the two approaches on benchmark $j30$

$j30$	$\|flex\|$	$\|fldt\|$	cpu	npc	mk
ISES + MAXCC$_{fldt}$	0.35	0.61	8.93	39.59	98.66
ISES + MAXCC$_{flex}$	0.38	0.56	8.85	43.47	98.73
ISES + MINID$_{fldt}$	0.45	0.63	8.65	30.95	98.64
ISES + MINID$_{flex}$	0.48	0.60	8.65	30.47	98.70
GRASP with MAXCC$_{fldt}$	0.39	0.62	6.46	39.13	105.43
GRASP with MAXCC$_{flex}$	0.41	0.59	6.54	40.40	105.65
GRASP with MINID$_{fldt}$	0.45	0.69	6.43	29.36	105.08
GRASP with MINID$_{flex}$	0.48	0.67	6.44	29.17	105.25

Table 6 Comparison between the two approaches on benchmark $j100$

$j100$	$\|flex\|$	$\|fldt\|$	cpu	npc	mk
ISES + MAXCC$_{fldt}$	0.11	0.53	46.03	56.17	434.47
ISES + MAXCC$_{flex}$	0.12	0.51	45.89	57.20	433.69
ISES + MINID$_{fldt}$	0.21	0.63	46.00	42.17	413.47
ISES + MINID$_{flex}$	0.21	0.62	46.05	42.20	413.69
GRASP with MAXCC$_{fldt}$	0.15	0.56	28.22	48.31	440.01
GRASP with MAXCC$_{flex}$	0.17	0.54	28.15	49.04	438.30
GRASP with MINID$_{fldt}$	0.20	0.65	27.85	38.77	435.79
GRASP with MINID$_{flex}$	0.21	0.63	27.75	38.84	437.46

the *fldt* metric. In this case, we have a difference in the quality values obtained: from 0.67 to 0.63. A different behavior is also observed in case of the GRASP variant: from 0.67 in Table 3 to 0.69. If the alternative chaining method MAXCC is used instead, the results obtained are much worse, especially in the case of ISES where the solutions are optimized according to the makespan (see the next section).

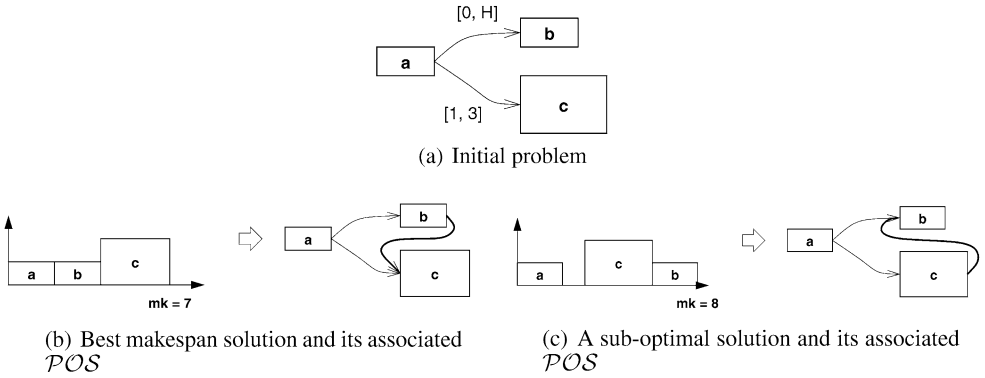
Table 6 shows the results obtained in the case of the benchmark $j100$. In this case, the differences between the two approaches are smoother than in the previous benchmark. In case of *flex*, we see that the ISES-based variants slightly outperform the GRASP ones. The opposite result is achieved in case of *fldt*: in this case we have 0.65 and 0.63, respectively, for the best GRASP variant and the best ISES-based variant. The difference in behavior observed on the $j100$ and $j30$ problem sets underscores the distinction mentioned earlier between the two benchmarks. In fact, despite the size of each instance, the benchmark $j100$ presents a set of instances that are simpler with respect to the $j30$, both in terms of the temporal network and resource usage.

5.3.1 Makespan versus robustness

The trade-off between makespan and robustness highlighted above is worth a comment. We have previously observed similar results when utilizing the ISES method to optimize the makespan value of the initial schedule and to increase the efficiency of the solving process. These results have consistently indicated that optimization of the makespan value tends to reduce the ability to obtain robust schedules, especially if the fluidity parameter is taken into account.

Figure 4 illustrates possible reasons why makespan optimization can have different effects with respect to flexibility and fluidity. Figure 4(a) presents a simple problem involving three activities that must be scheduled on a single resource. The activities are ordered according to the constraints in the figure, i.e., between a and b there is a simple precedence constraint while between a and c the constraint specifies a time window $[1, 3]$, i.e., c must start sometime between 1 and 3 time units after activity a ends. Furthermore, the size of each activity describes both its duration (the width) and its resource need (the height). Therefore, both a and b require one resource unit for two time units while c has a duration of 3 and a resource requirement of 2. To complete the description of the problem, the resource capacity is 2.

Figure 4(b) and 4(c) show two fixed times solutions with their associated partial order schedule (the darker arrows represent the additional constraints necessary to obtain a flexible solution). Note that in this case there is a single $\mathcal{POS}$ for each fixed-times solution: in fact, each of the two proposed solutions gives a complete linearization of the activities. The two schedules have different makespan values 7 and 8, respectively.



(a) Initial problem

(b) Best makespan solution and its associated $\mathcal{POS}$

(c) A sub-optimal solution and its associated $\mathcal{POS}$

Fig. 4 Example: dependency between makespan and robustness

Let us first consider the flexibility parameter. Examining the two partial order schedules (on the right hand side of Fig. 4(b) and 4(c)) it is possible to notice that both have the same *flex* value. In fact, the *flex* value is zero in both cases because there is no pair of activities that is not reciprocally ordered.

The fluidity metric considers the slack value between any pair of activities, that is, the minimum and maximum distance between them. For the pair (a, c), both solutions yield the same value, $dist(a, c) = [1, 3]$, which stems from the time window constraint defined between the two activities. For the other pair (a, b), however, we have two different values: $dist(a, b) = [0, 1]$ in the case in Fig. 4(b) and $dist(a, b) = [4, H - 2]$ in the case in Fig. 4(c). This clearly shows that the fluidity value for the non-optimal solution is greater than the optimal solution in Fig. 4(b). The same behavior can be found for the pair (b, c), where we have $dist(b, c) = [0, 1]$ for the case in Fig. 4(b) while for the case in Fig. 4(c) $dist(c, b) = [0, H - 6]$.

The problem is that in the schedule with the optimal makespan (Fig. 4(b)) the activity b is "caged in" by the other two activities. Therefore, the time window constraint defined between a and c has the effect of limiting the flexibility of activity b. Furthermore, since the capacity of the resource is equal to the requirement of c there is no chaining method able to overcome the problem. This circumstance is due to the presence of maximum distance constraints (or window constraints), like in the case of RCPSP/max. In fact, if the maximum constraint between a and c did not exist, the activity b would have the ability to move back and forth in a larger interval, resulting in a more flexible solution.

5.4 Extending the analysis: stability

In Sect. 5.1, we have introduced two metrics: flexibility (*flex*) and fluidity (*fldt*). These have been used to obtain an assessment of the qualities of the produced solutions ($\mathcal{POS}$s) in terms of their ability to absorb unexpected events. In fact, in the case of a very complex scheduling problem (like RCPSP/max), the ability to respond to unforeseen events without requiring a new scheduling phase becomes of fundamental worth.

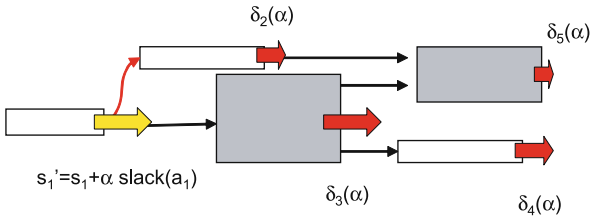


Fig. 5 Stability evaluation: the impact of the shift $\alpha slack(a_i)$ of the start-time of a_i over the remaining activities is represented by the values $\delta_j(\alpha)$

Nevertheless, when a solution is repaired it is possible that several changes will be made; such an approach produces an instability of the solution and nervousness in the execution. Furthermore, preserving a stable solution also has a different outcome; with respect to a given partial order schedule, maintaining a stable solution is equivalent to retaining as many solutions as possible in the POS.

For these reasons, we present a further analysis of our methods for constructing $\mathcal{POS}$s. Specifically, we introduce an additional metric to characterize the stability of the produced solutions. For example, let us suppose that the start time s_i of the activity a_i is delayed by Δ_{in}. An interesting aspect is the average delay of the other activities Δ_{out}. To evaluate the stability of a solution we consider a single type of modification event (see Fig. 5): the start time s_i of a single activity a_i with a *window* of possible start times $slack(a_i) = [est(a_i), lst(a_i)]$ is increased to a value $s_i + \alpha slack(a_i)$, where $0 \leq \alpha \leq 1$. A more operative definition of stability is given by the following formula:

$$stby(\alpha) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{\delta_j(\alpha)}{slack(a_i)}, \tag{9}$$

where the stability $stby(\alpha)$ is defined as the average value $\frac{\delta_j(\alpha)}{slack(a_i)}$ over all pairs (a_i, a_j), with $a_i \neq a_j$, when an in-

Table 7 Stability evaluation on the benchmark set $j30$

%α	1	2	4	8	16	20	25	50	75	100
chn	0.05	0.09	0.13	0.17	0.21	0.22	0.23	0.27	0.29	0.30
$rndm_{fldt}$	0.05	0.08	0.12	0.16	0.20	0.21	0.22	0.25	0.28	0.29
$rndm_{flex}$	0.05	0.08	0.11	0.15	0.19	0.21	0.22	0.26	0.28	0.29
MAXCC_{fldt}	0.05	0.08	0.11	0.15	0.18	0.19	0.20	0.22	0.24	0.25
MAXCC_{flex}	0.05	0.08	0.11	0.15	0.18	0.19	0.21	0.24	0.26	0.27
MINID_{fldt}	0.03	0.06	0.09	0.12	0.15	0.16	0.16	0.19	0.20	0.21
MINID_{flex}	0.03	0.06	0.08	0.12	0.15	0.16	0.17	0.20	0.22	0.23

Table 8 Stability evaluation on the benchmark set $j100$

%α	1	2	4	8	16	20	25	50	75	100
chn	0.10	0.14	0.20	0.26	0.31	0.33	0.35	0.39	0.41	0.43
$rndm_{fldt}$	0.09	0.13	0.18	0.25	0.29	0.31	0.33	0.37	0.38	0.41
$rndm_{flex}$	0.09	0.14	0.19	0.25	0.30	0.32	0.34	0.38	0.39	0.42
MAXCC_{fldt}	0.09	0.13	0.18	0.23	0.28	0.30	0.31	0.35	0.37	0.38
MAXCC_{flex}	0.09	0.13	0.18	0.24	0.30	0.31	0.33	0.37	0.39	0.40
MINID_{fldt}	0.05	0.09	0.13	0.17	0.21	0.22	0.23	0.27	0.28	0.29
MINID_{flex}	0.05	0.08	0.13	0.17	0.22	0.23	0.24	0.27	0.29	0.30

crease of start time $\alpha slack(a_i)$ is performed on each activity start time separately. We observe that the single value $\frac{\delta_j(\alpha)}{slack(a_i)}$ represents the relative increment of the start time of the activity a_j (the absolute value is $\delta_j(\alpha)$) when the start time of the activity a_i is increased to the value $s_i + \alpha slack(a_i)$. For this reason, the lower the value of $stby(\alpha)$ the better the quality of the solution.

Table 7 and Table 8 present the results obtained evaluating stability, respectively, on the benchmark sets $j30$ and $j100$. The data represent the normalized values of the stability metric ($|stby(\alpha)|$) for different values of α (in percentage). The normalization is obtained considering both a lower bound and an upper bound for the metric. The lower bound *LB* is obtained relaxing the resource constraints of the single instances and evaluating the stability of such a temporal net.[9] Conversely, an upper bound *UB* for the stability values is represented by the α value, i.e., $stby(\alpha) \leq \alpha$. Then the following formula is considered:

$$0 \leq \left|stby(\alpha)\right| = \frac{stby(\alpha) - LB}{UB - LB} \leq 1.$$

The results[10] in Table 7 show that the solutions obtained using the MINID procedures result in a better behavior across all values of α. Therefore, minimization of the number of chain interdependencies reduces the sensitivity of the solution. Indeed, for $\alpha = 1$, while the MAXCC procedures are able to obtain at most a reduction of the $|stby(\alpha)|$ value of about 17%, it is possible to obtain an improvement of 30% using the enhanced method. It is worth noting that the best results, for $\alpha = 1$, for both the procedures, MAXCC and MINID, are obtained when the *fldt* parameter is optimized. Moreover, the improvement is observed across all the values of α: for instance, for $\alpha = 0.04$ we have a reduction of about 38% (from 0.13 to 0.08).

The ability of MINID to produce stable partial order schedules is confirmed analyzing the results for the benchmark set $j100$ (see Table 8). Also in this case, in fact, we see that the MAXCC procedure obtains an improvement of about 11% for $\alpha = 1$, while the MINID procedure has a reduction of about 33%.

6 Related work

The robustification approach described in this paper is rooted in early research in AI planning and scheduling in different ways. The closest connection is with research on flexible solutions for constraint-based scheduling (Smith and Cheng 1993; Cheng and Smith 1994). In these approaches, a temporal graph is used to retain temporal flexibility in the final schedule, which can later be exploited when there is uncertainty associated with executing activities. Unfortunately,

[9] Once the resource constraints are removed the resulting graph will constitute a partial order schedule.

[10] The values presented represent the average stability values of the solutions obtained through the several methods.

exploiting this flexibility might often lead to other kinds of inconsistencies. This is due to the fact that not all allocations in time allowed by the flexibility are also resource consistent, and there might be many assignments to time points which, though temporally consistent, could trigger resource conflict in the solution. The idea of a $\mathcal{POS}$ has been actually designed to overcome this limitation.

Another natural ancestor of a $\mathcal{POS}$ is the concept of partial order plan used in AI planning that, similarly to $\mathcal{POS}$, has the idea of synthesizing more than one solution in a compact representation—see as examples Chapman (1987), Kambhampati et al. (1995). Indeed, in Policella et al. (2007), we considered a pure least commitment method to synthesizing partial order schedules, consistent with the typical refinement search paradigm. Such an algorithm computes upper and lower bounds on resource usage across all possible executions according to the exact computations 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. That paper shows how a very basic $\mathcal{POS}$ computation based on generalization from a single solution, outperforms the least commitment synthesis. For a similar result in planning see Do and Kambhampati (2003).

Other works have considered the concept of flexible solutions with different interpretations. For instance, Wu et al. (1999) point out how a complete resolution of a scheduling problem can limit its flexibility. They instead propose a partial resolution of the problem which entails a decomposition of the problem into a series of ordered subproblems. This method postpones resolution of some resource conflicts in the schedule until it is being executed. This consists of a first, off-line, step in which a selected subset of conflicts are solved; in the following on-line phase, the remaining scheduling decisions are dynamically considered. Artigues et al. (2005) present a similar approach based on the ordered group assignment (OGA) representation; this is defined in terms of a sequence of groups for each machine, where the activities within a group are totally permutable. Therefore, during execution phase, the decision maker is able to choose any activity in the group. Solving some of the conflicts during the execution can be useful because it allows the actual evolution of the schedule (e.g., activity delays, resource unavailability) to solve the pending conflicts. However, this aspect may turn out to be a limitation when more complex scheduling problems are considered. In fact, in this case heavy computational cost can preclude the possibility of keeping pace with the execution (introducing further delays). It is worth remarking that, with respect to these two works, we address a very complex problem, RCPSP/max, which is not polynomial-time solvable, and hence it enhances the need for flexible solutions.

In Rasconi et al. (2008), a complementary $\mathcal{POS}$ evaluation is given. In particular, $\mathcal{POS}$s are evaluated against other methods for reactive adjustment of a solution. These papers discuss alternative approaches to the execution, monitoring and repair of scheduling solutions. Two alternatives to the classical fixed time schedule are analyzed, namely flexible schedules, containing a single point solution, and partial order schedules. The experiments shed light on many possible trade-offs that must be faced, depending on the conditions of execution as well as on the particular aspects that must be emphasized. When fast responsiveness and stability are primary concerns, the $\mathcal{POS}$ is extremely efficient because it rarely requires rescheduling in comparison to flexible schedules.

Partial order schedules have also been used to achieve different aims. For instance, in Godard et al. (2005), Laborie and Godard (2007) $\mathcal{POS}$s are used to overcome the natural lack of flexibility of fixed time solutions and to apply a large neighborhood search schema. In Policella et al. (2005), a Two-Step approach is used to obtain high quality solutions for a particular scheduling problem where each activity to be scheduled has a duration-dependent quality profile, and activity durations must be determined that maximize overall quality within given deadline and resource constraints. Also in this case the $\mathcal{POS}$ concept is used to overcome the lack of flexibility of fixed time solutions: once a $\mathcal{POS}$ is obtained it is possible to exploit its temporal flexibility (to obtain high quality solutions) by applying an LP solver. These works show how the $\mathcal{POS}$ concept can also be efficiently applied to other scheduling problems.

We conclude by remarking that a $\mathcal{POS}$ can be functional to the integration of planning and scheduling in dealing with the robustness and stability issues discussed in this paper. The integration of planning and scheduling is often seen as a key feature for solving real-world problems. Several planning architectures defined over the past two decades have included aspects from both planning and scheduling theories (Currie and Tate 1991; Chien et al. 1998; Laborie and Ghallab 1995). Also the recent international planning competitions (IPC) have considered this integration as a direction to follow and specific extended features appear in the most recent release of the PDDL language (Fox and Long 2003). Indeed, a $\mathcal{POS}$ can provide a uniform and *symmetrical* structure to the causal graph used in planning for representing resource and temporal constraints within the same graph structure. We believe such common structure for representing the main elements of the integrated problem, might promote the definition of effective execution and re-planning algorithms. For now, we observe that the issues proposed in this work about robustness and stability in scheduling domains already have their counterparts within the planning literature. In particular, the definition of solution robustness and its metrics (Fox et al. 2006b) and the problem of providing efficient re-planning algorithms (Fox et al. 2006a) which guarantee a stable solution are emerging

as relevant research themes for moving an abstract planning solution to a real word scenario.

7 Conclusions

In this paper, we have considered the problem of transforming a fixed-times schedule into a *Partial Order Schedule* ($\mathcal{POS}$) to enhance its robustness. In Policella et al. (2007), we showed how a two-step procedure—find a solution then make it robust—represents an efficient way to generate flexible, robust schedules. Following this result we introduced a new schema, *Solve-and-Robustify*, where a fixed times schedule is generated in stage one, and then a procedure referred to as chaining is applied to transform this fixed-times schedule into a Partial Order Schedule ($\mathcal{POS}$). In the present work we have significantly extended this approach to synthesizing $\mathcal{POS}$s, introducing both an iterative improvement schema and two heuristics for decreasing possible "blocking among chains". We provided a deeper discussion of chaining procedures and we evaluated their scalability over larger RCPSP/max benchmarks. Finally, we investigated further possibilities to broaden the search for partial order schedules by evaluating two more sophisticated Solve-and-Robustify approaches.

We focused specifically on generating $\mathcal{POS}$s via *chaining procedures* where activities competing for the same resources are linked into precedence chains. The paper pointed out two basic properties: (1) a given $\mathcal{POS}$ can always be generated by a chaining process, and (2) this process is makespan preserving with respect to an input schedule. As a consequence, issues of minimizing schedule makespan and maximizing schedule robustness can be addressed sequentially in a two-step scheduling procedure. Starting from the first property, we considered the possibility of producing $\mathcal{POS}$s with better robustness and stability properties through more extended search in the space of $\mathcal{POS}$s. In particular, two iterative sampling chaining procedures are proposed: each taking into account structural properties of robust $\mathcal{POS}$s to heuristically bias chaining decisions. To evaluate these procedures, we used metrics for assessing the robustness and stability of a generated $\mathcal{POS}$.

Experimental results were shown to confirm the effectiveness of the extended search approach. In general, consideration of $\mathcal{POS}$s emphasizes the presence of synchronization points as obstacles to flexibility, and this fact can be exploited to generate $\mathcal{POS}$s with good robustness properties. Also we have noticed an interesting trade-off between solution makespan and solution flexibility. In practice, the use of more optimized initial schedule biases the robustify phase against the construction of flexible partial order schedules. In fact, the tightness (makespan) of the initial solution can preclude the achievement of good flexible solutions. Therefore, while the robustify step does not degrade the results obtained in the solve phase, some characteristics of the fixed-times solutions may lower the robustness of the final partial order schedules.

Acknowledgements Nicola Policella is currently supported by a Research Fellowship of the European Space Agency, Human Spaceflight and Explorations Department. Amedeo Cesta and Angelo Oddi's work is partially supported by MIUR (Italian Ministry for Education, University and Research) under project VINCOLI E PREFERENZE (PRIN), CNR under project RSTL (Funds 2007) and ESA (European Space Agency) under the APSI project. Stephen F. Smith's work is supported in part by the National Science Foundation under contract #9900298, by the Department of Defense Advanced Research Projects Agency under contract # FA8750-05-C-0033 and by the CMU Robotics Institute.

References

Aloulou, M. A., & Portmann, M. C. (2005). An efficient proactive-reactive scheduling approach to hedge against shop floor disturbances. In G. Kendall, E. Burke, S. Petrovic, & M. Gendreau (Eds.), *Multidisciplinary scheduling: theory and applications* (pp. 223–246). New York: Springer.

Artigues, C., Billaut, J. C., & Esswein, C. (2005). Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, *165*(2), 314–328.

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

Cesta, A., & Oddi, A. (2001). *Algorithms for dynamic management of temporal constraints networks* (Tech. rep.). ISTC-CNR, Institute for Cognitive Science and Technology, Italian National Research Council.

Cesta, A., Oddi, A., & Smith, S. F. (1998). Profile based algorithms to solve multiple capacitated metric scheduling problems. In *Proceedings of the 4th international conference on artificial intelligence planning systems, AIPS-98* (pp. 214–223).

Cesta, A., Oddi, A., & Smith, S. F. (2002). A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, *8*(1), 109–136.

Chapman, D. (1987). Planning for conjuctive goals. *Artificial Intelligence*, *32*(3), 333–377.

Cheng, C., & Smith, S. F. (1994). Generating feasible schedules under complex metric constraints. In *Proceedings of the 12th national conference on artificial intelligence, AAAI-94* (pp. 1086–1091). Menlo Park: AAAI Press.

Chien, S. A., Muscettola, N., Rajan, K., Smith, B. D., & Rabideau, G. (1998). Automated planning and scheduling for goal-based autonomous spacecraft. *IEEE Intelligent Systems*, *13*(5), 50–55.

Currie, K., & Tate, A. (1991). O-plan: The open planning architecture. *Artificial Intelligence*, *52*(1), 49–86.

Do, M. B., & Kambhampati, S. (2003). Improving temporal flexibility of position constrained metric temporal plans. In *Proceedings of the 13th international conference on automated planning & scheduling, ICAPS'03* (pp. 42–51).

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, *20*, 61–124.

Fox, M., Gerevini, A., Long, D., & Serina, I. (2006a). Plan stability: replanning versus plan repair. In *Proceedings of the 16th international conference on automated planning and scheduling, ICAPS 06* (pp. 212–221).

Fox, M., Howey, R., & Long, D. (2006b). Exploration of the robustness of plans. In *Proceedings of the 21st national conference on artificial intelligence, AAAI 06* (pp. 834–839).

Godard, D., Laborie, P., & Nuitjen, W. (2005). Randomized large neighborhood search for cumulative scheduling. In *Proceedings of the 15th international conference on automated planning and scheduling, ICAPS 2005* (pp. 81–89).

Kambhampati, S., Knoblock, C. A., & Yang, Q. (1995). Planning as refinement search: a unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence*, *76*(1–2), 167–238.

Kolisch, R., Schwindt, C., & Sprecher, A. (1998). Benchmark instances for project scheduling problems. In J. Weglarz (Ed.), *Project scheduling—recent models, algorithms and applications* (pp. 197–212). Boston: Kluwer Academic.

Laborie, P., & Ghallab, M. (1995). Planning with sharable resource constraints. In *Proceedings of 14th international joint conference on artificial intelligence, IJCAI-95* (pp. 1643–1651).

Laborie, P., & Godard, D. (2007). Self-adapting large neighborhood search: application to single-mode scheduling problems. In *Proceedings of 3rd multidisciplinary international scheduling conference: theory and applications, MISTA-07*.

Leus, R., & Herroelen, W. (2004). Stability and resource allocation in project planning. *IIE Transactions*, *36*(7), 667–682.

Muscettola, N. (2002). Computing the envelope for stepwise-constant resource allocations. In *Lecture notes in computer science: Vol. 2470. Principles and practice of constraint programming, 8th international conference, CP 2002* (pp. 139–154). Berlin: Springer.

Policella, N., Oddi, A., Smith, S. F., & Cesta, A. (2004a). Generating robust partial order schedules. In M. Wallace (Ed.), *Lecture notes in computer science: Vol. 3258. Principles and practice of constraint programming, 10th international conference, CP 2004* (pp. 496–511). Berlin: Springer.

Policella, N., Smith, S. F., Cesta, A., & Oddi, A. (2004b). Generating robust schedules through temporal flexibility. In *Proceedings of the 14th international conference on automated planning & scheduling, ICAPS'04* (pp. 209–218).

Policella, N., Wang, X., Smith, S. F., & Oddi, A. (2005). Exploiting temporal flexibility to obtain high quality schedules. In *Proceedings of the twentieth national conference on artificial intelligence, AAAI-05* (pp. 1199–1204).

Policella, N., Cesta, A., Oddi, A., & Smith, S. F. (2007). From precedence constraint posting to partial order schedules. *AI Communications*, *20*(3), 163–180.

Rasconi, R., Cesta, A., & Policella, N. (2008). Validating scheduling approaches against executional uncertainty. *Journal of Intelligent Manufacturing* (in press).

Resende, M., & Ribeiro, C. (2002). Greedy randomized adaptive search procedures. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 219–249). Dordrecht: Kluwer Academic.

Smith, S. F., & Cheng, C. (1993). Slack-based heuristics for constraint satisfactions scheduling. In *Proceedings of the 11th national conference on artificial intelligence, AAAI-93* (pp. 139–144). Menlo Park: AAAI Press.

Wu, S. D., Beyon, E. S., & Storer, R. H. (1999). A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, *47*(1), 113–124.