Schedule robustness through *Solve-and-Robustify*: generating flexible schedules from different fixed-time solutions

Nicola Policella, Amedeo Cesta and **Angelo Oddi**
Planning & Scheduling Team
Institute for Cognitive Science and Technology - CNR
Rome, Italy
name.surname@istc.cnr.it

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

Abstract

In previous works the authors have defined *Solve-and-Robustify* an original two-step procedure to generate flexible solutions, or partial order schedules, for scheduling problems. The partition in two steps — first find a solution then make it robust — not only represents a way to generate flexible, robust schedules but is also an alternative to achieve good quality solutions. This paper extends the analysis of this paradigm investigating the effects of using different start solutions as a baseline to generate partial order schedules. Two approaches are compared: the first based on the construction of flexible schedules after performing an iterative improvement phase focused on makespan optimization, the second that selects the best partial order schedule considering different fixed-time schedules as starting points. The paper experimentally shows how the characteristics of the fixed-time solutions may lower the robustness of the final partial order schedules and discusses motivations for such behavior.

Introduction

In previous works (Policella *et al.* 2004b; 2004a) these authors show how a two step procedure — find a solution then make it robust — represents a way to generate flexible, robust schedules and, to improve the solution quality in several directions. Under this scheme, a feasible fixed-time schedule is generated in stage one (in particular an early start times solution is identifies), and then, in the second stage, a procedure referred to as *chaining* is applied to transform this fixed-time schedule into a *Partial Order Schedule*, or $\mathcal{POS}$. The same works explain why a $\mathcal{POS}$ represents a robust solution.

The common thread underlying the "chained" representation of the schedule is the characteristic that activities which require the same resource units are linked via precedence constraints into precedence chains. Given this structure, each constraint becomes more than just a simple precedence. It also represents a *producer-consumer* relation, allowing each activity to *know* the precise set of predecessors which will *supply* the units of resource it requires for execution. In this way, the resulting network of chains can be interpreted as a flow of resource units through the schedule; each time an activity terminates its execution, it passes its resource unit(s) on to its successors. It is clear that this representation is robust if and only if there is temporal slack that allows chained activities to move "back and forth". Concepts similar to chaining have also been used elsewhere: for example, the Transportation Network introduced in (Artigues & Roubellat 2000), and the Resource Flow Network described in (Leus & Herroelen 2004) are based on equivalent structural assumptions.

This paper addresses an aspect not explored in previous works: how different start schedules influence the whole process of identifying partial order schedules. This analysis is presented here describing two different combinations of a constraint-based solver with our best chaining algorithm from previous works: the first combination schema constructs flexible schedules after an iterative sampling optimization procedure, while the second iterates both solve and robustify considering different fixed-time schedules as starting points and selecting the best partial order schedule found.

The paper introduces first the basic concepts of schedule robustness and Partial Order Schedules, then describes the two step approach to $\mathcal{POS}$ synthesis. The new analysis is then introduced by describing the broadened search procedures, an experimental evaluation and a detailed discussion. Some conclusions end the paper.

Scheduling with Uncertainty and Partial Order Schedules

The usefulness of schedules in most practical scheduling 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 unforeseen events quickly invalidate the schedule's predictive assumptions and bring into question the continuing validity of the schedules's prescribed actions. The lifetime of a schedule tends to be very short, and hence its optimizing advantages are generally not realized. For instance, let us consider the example in Fig. 1 that shows the allocation of three different activities on a binary resource. According to quite common practice in scheduling, a solution associates an exact start and end time to each activity.

(a) Initial allocation of three activities on a binary resource

(b) Allocation after an activity lasts longer

Figure 1: Brittleness of a fixed-time schedule

Such solution may exhibit a high degree of brittleness, for instance, as shown in Fig. 1(b), when the first activity lasts more than expected a conflict in the usage of the machine is immediately generated because of the fixed start-time for the activities.

An alternative approach consists of adopting a graph formulation of the scheduling problem, wherein activities competing for the same resources are simply ordered to establish resource feasibility, it is possible to produce schedules that retain temporal flexibility where allowed by the problem constraints. In essence, such a "flexible schedule" encapsulates a set of possible fixed-time schedules, and hence is equipped to accommodate some amount of the uncertainty at execution time.

Following this intuition we have introduced the definition of *Partial Order Schedules*, or $\mathcal{POS}$s (Policella 2005). It consist in a set of feasible solutions for the scheduling problem that can be represented in a compact way by a temporal graph, that is, a graph in which any activity is associated to a node and temporal constraints that define the order in which such activities have to be executed.

To provide a more formal definition of a $\mathcal{POS}$ we use the activity on the node representation: given a 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 and 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 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 the possible resource conflicts. Given these concepts, a *Partial Order Schedule* is defined as follows:

Definition 1 (Partial Order Schedule) *Given a scheduling problem P and the associated representing 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 $G_{POS}(V_P, E_P \cup E_R)$.*

In practice a $\mathcal{POS}$ is a set of partially ordered activities such that any possible complete activity allocation that is consistent with the initial partial order is also a resource and time feasible schedule.

It is worth noting that a partial order schedule provides an immediate opportunity to reactively respond to some of the possible external changes by simply propagating their effects over the "graph", by using a polynomial time computation. In fact the augmented duration of an activity, as well as a greater release time, can be modeled as a new temporal constraint to post on the graph. It is also important to note that, even though the propagation process does not consider the consistency with respect the resource constraints, it is guaranteed to obtain a feasible solution by definition of $\mathcal{POS}$s. Therefore a partial order schedule provides a mean to find a new solution and ensures its fast computation.

RCPSP/max. This work considers the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max (Bartusch, Mohring,

& Radermacher 1988) as the reference problem. The basic entities of this problem are a set of *activities* denoted by $V = \{a_1, a_2, \dots a_n\}$. Each activity has a fixed *processing time*, or *duration*, p_i and must be scheduled without preemption.

A *schedule* is an assignment of start times to activities $a_1, a_2, \dots a_n$, i.e. a vector $S = (s_1, s_2, \dots, s_n)$ where s_i denotes the start time of activity a_i. The time at which activity a_i has been completely processed is called its *completion time* and is denoted by e_i. Since we assume that processing times are deterministic and preemption is not permitted, completion times are determined by $e_i = s_i + p_i$. Schedules are subject to both *temporal* 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, $l_{ij}^{min} \leq s_j - s_i \leq l_{ij}^{max}$ where l_{ij}^{min} and l_{ij}^{max} are the minimum and maximum time lag of activity a_j relative to a_i. A schedule $S = (s_1, s_2, \dots, s_n)$ is *time feasible*, if all inequalities given by the activity precedences/time lags and durations hold for start times s_i. During their processing, activities require specific resource units from a set $R = \{r_1, r_2, \dots r_m\}$ of resources. Resources are *reusable*, i.e. they are released when no longer required by an activity and are then available for use by another activity. Each activity a_i requires of the use of req_{ik} units of the resource r_k during its processing time p_i. Each resource r_k has a limited capacity of 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. $\sum_{s_i \leq t < e_i} req_{ik} \leq c_k$. A schedule S is called *feasible* if it is both time and resource feasible.

Metrics to Compare Partial Order Schedules. As described before, a single $\mathcal{POS}$ represents a set of temporal solutions that are also resource feasible. This set of schedules provides a mean for tolerating some amount of execution uncertainty. When an unexpected event occurs (e.g., a start time delay), the temporal propagation mechanism (a polynomial time calculation) can be applied to update the start times of all activities and, if at least one temporal solution remains viable, produces a new partial order schedule. Therefore, it is intuitive that the quality of a certain $\mathcal{POS}$ is tightly related to the set of solutions that it can represent. In fact the greater the number of solutions, the greater is the expected ability of facing scheduling uncertainty. Furthermore, an other aspect to consider in the analysis of the solutions clustered into a partial order schedule is the distribution of such alternatives over all the activities. Such distribution will be the result of the configuration given by the constraints present in the solution. For this reason it is necessary to introduce metrics that consider such aspects.

A first measure, $flex$, is taken from (Aloulou & Portmann 2003). This measure counts the *number of pairs of activities in the solution which are not reciprocally related by simple precedence constraints*. This provides a first analysis of the configuration of the solution. The rationale is that when two activities are not related it is possible to move one without moving the other one. Hence, the higher the value of $flex$ the lower the degree of interaction among the activities. It is worth noting that a limitation of the $flex$ metric consists in being able to give only a qualitative evaluation of the solution because it counts only existence/not-existence of a direct ordering relation. This may be sufficient for scheduling problem with no time lag constraints like the one used in (Aloulou & Portmann 2003), but is rather limited to describe flexibility in RCPSP/max where it is necessary to integrate this flexibility measure with some factor that takes into account the quantitative aspects of the temporal problem (or solution).

A possible metric that satisfies this requirement is defined in (Cesta, Oddi, & Smith 1998). It requires the presence of a fixed time horizon for the termination of all the activities. In order to compare two or more $\mathcal{POS}$s we bound any partial order schedule to have a finite number of solutions. Then the metric 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):

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

where H is the temporal horizon of the problem, 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[1]. We use this metric to characterize the *fluidity* of a solution, i.e., the ability to use flexibility to absorb a temporal variation in the execution of activities. It also considers that a temporal variation concerning an activity is absorbed by the temporal flexibility of the solution instead of generating deleterious domino effect (the higher the value of $fldt$, the less the risk, i.e., the higher the probability of localized changes).

Solve and Robustify

In (Cesta, Oddi, & Smith 1998) a two-stage approach to generating a flexible schedules is introduced as one possibility for robust schedules. Under this scheme a feasible fixed-time schedule is first generated in stage one, and then, in the second stage, a procedure referred to as *chaining* is applied to obtain a robust solution. In this second step, fixed-time commitments are converted into a sequences (chains) of activities to be executed by various resources.

In a recent paper, (Policella *et al.* 2004b), this approach has been generalized to RCPSP/max. These results establish the basic viability of a chaining procedure. At the same time, the procedure used in this work was developed simply to provide a means of transforming a given schedule into a $\mathcal{POS}$; no attention was given to the potential influence of the chaining procedure itself on the properties exhibited by the final, flexible solution. For these reasons, in (Policella *et al.* 2004a) the authors examine the problem of generating $\mathcal{POS}$s from the broader perspective of producing flexible schedules with good robustness properties, and investigate the design of informed chaining procedure that exploit

[1]In (Cesta, Oddi, & Smith 1998) this metric was defined as robustness of a solution, RB.

knowledge of these properties to increase the robustness of the final $\mathcal{POS}$. This last work ended up introducing an iterative improvement schema for the a randomized chaining algorithm. In (Policella *et al.* 2004b) the idea of coupling two modules to obtain partial order schedules has been proposed. The ingredients of this approach are a solving method (to get a fixed-time schedule) and a robustify phase that feeds flexibility into the original solution – such flexibility directly face execution uncertainty.

A sketchy representation of the *Solve-and-Robustify* approach can be seen, Fig. 2(b), as an open loop cascade of a "solver" and a "robustify" modules. The subsequent work in (Policella *et al.* 2004a) can be represented as a first loop with respect to the simple cascade by applying an iterative improvement cycle around the robusty phase, Fig. 2(b). Iterative improvement is obtained by randomizing the basic procedure to obtain different search paths from different restarts[2].

The ESTA greedy solver. As a solver we use here the *Precedence Constraint Posting* greedy schema called ESTA described in (Cesta, Oddi, & Smith 1998; 2002). This proceeds analyzing the infinite capacity representation of a RCPSP/max problem where the temporal constraint are satisfied and the resource profiles contain violations. More precisely, a violation or *resource contention peak* consists in a set of activities that are executed at the same time and that require an amount of resource greater than the resource capacity. Then the solving process posts further precedence constraints to remove these peaks. The selection of the new constraint is accomplished by using three alternative heuristics; a first, simple strategy consists in considering all the pairs of activities in a peak. Other two approaches use the MCS, minimal conflict sets, where an MCS is a set of resource conflicting activities such that any of its proper subset is consistent. To avoid the complexity of computing all the MCSs two polynomial sampling approaches are used: *linear* and *quadratic*. The use of the three different heuristics gives rise to three different variants of the ESTA algorithm.

Iterative chaining. In our current approach the robustification phase is carried out through chaining. A chaining procedure transforms a feasible fixed-time solution into a $\mathcal{POS}$ by dispatching activities to specific resource units[3]. Since choices can be made as to how to dispatch activities to resource units, it is possible to generate different $\mathcal{POS}$s from the same starting solution, and these different $\mathcal{POS}$s can be expected to have different robustness properties.

In (Policella *et al.* 2004a) iterative sampling is used to explore this set of solutions. Randomization is added to obtain a different solution at each iteration and, in so doing, to generate a sequence of $\mathcal{POS}$s starting from the same initial schedule. The $\mathcal{POS}$s are evaluated with respect to a specific

(a) basic solve and robustify

(b) improved chaining

Figure 2: Previous work

[2]In the figure, a module containing a randomized procedure is labeled with a star (e.g., *Robustify**).

[3]Note that this procedure is required to enable schedule execution.

robustness measure returning the best one found. Additionally, different heuristics have been explored that change the effectiveness of chaining. In this paper we use the more effective one called MINID (*minimizing interdependencies*) in (Policella 2005).

The heuristic MINID takes into account existing ordering relations with those activities already allocated in the chaining process in order to minimize possible links among pairs of chains which will degrade the flexibility of the solution. In this procedure, the allocation of an activity a_i on the chains of a multi-capacitive resource r_j proceeds according to the following steps:

(1) collect in the set P_{a_i}, the chains k belonging to r_j, for which their last element, $last_k$, is already ordered with respect to the activity a_i, i.e., $last_k \prec a_i$;

(2) 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;

(3) a constraint $a_k \prec a_i$ is posted, where $a_k = last_k$;

(4) 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, $\bar{C_{a_k}}$;

(5) to satisfy all remaining resource requirements, a_i is allocated first to chains belonging to the first subset, $k' \in C_{a_k}$ and,

(6) 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, $k'' \in \bar{C_{a_k}}$.

Some properties of chaining. Why are we interested in the use of the chaining approach? Previous works have found several interesting properties.

First, any partial order schedule can be obtained by a chaining procedure (Policella *et al.* 2004a, Theorem 1). This result allows to restrict the attention, without loss of generality, to the set of $\mathcal{POS}$s generated by a chaining procedure.

Second, given a schedule S, the makespan of the earliest solution[4] of the partial order schedule generated through chaining, $\mathcal{POS}_S$, is not greater than the makespan of the input solution. Thus, the makespan of a solution is preserved by the chaining-based robustification phase.

Additionally, previous work has identified structural properties for the effectiveness of chaining in RCPSP/max. In particular, the analysis of solutions obtained via chaining has brought out the presence of *synchronization points* which tend to degrade solution flexibility. These stem from the presence of activities which require multiple resource units and from precedence constraints between activities allocated on different chains. In fact, each of these aspects will mutually constrain two (or more), otherwise independent processes.

[4]The solution in which each activity is allocated at the earliest start time defined by the partial order schedule.

Finally, an interesting aspect of the procedure is that it can be coupled with any fixed-time schedule generation procedure. Independently on the fact that we are using our own constraint-based approach to solving RCPSP/max, the robustify step can be applied to other approaches to solve the same scheduling problem like (Dorndorf, Pesch, & Phan-Huy 2000; Smith & Pyle 2004; Cicirello & Smith 2004).

Generating flexible schedules from different initial solutions

In previous work we have done no investigation on the effects that different fixed-time initial solutions may have to the final synthesized $\mathcal{POS}$. This complementary analysis is contributed by this paper. In particular we extend our schema as shown in Fig. 3 where dashed lines highlight the new steps: (a) understanding the influence on $\mathcal{POS}$s of a much broader search for a better makespan before the robustify step; (b) considering the whole solve-and-robustify open loop within a meta-heuristic schema. For the solving phase we draw on our own CSP solvers for RCPSP/max.

The iterative sampling procedure

The first approach we investigate in shown in Fig. 3(a). Instead of the simple ESTA greedy solver, we insert the ISES iterative improvement schema that is based on a randomized ESTA (hence *Solve** in the figure). The randomized version of ESTA is obtained by doing a pseudo-random selection of the decision. Conflict selection is performed not picking the best ranked conflict but choosing randomly within an acceptance band. This technique, introduced first in (Oddi & Smith 1997), eliminates heuristic bias when the discrimination power of the estimator is low. The whole approach is called ISES^{iC} because in practice it joins the ISES optimizer with the iterative chaining that uses the MINID method.

The grasp-like procedure

The second approach introduced in this paper is shown in Fig. 3(b). It follows a typical GRASP meta-heuristic schema (Resende & Ribeiro 2002). The idea is to use the randomized ESTA to create different start schedule with the greedy solver then apply the iterative chaining exploration (hence the GRASP^{iC} name). This whole schema is iterated until a termination criterion is met. Based on a robustness metric the best $\mathcal{POS}$ found is returned.

A further remark

In Fig. 4 we underscore graphically the different ways the iterative procedure and the GRASP-like follow through the search space. They both start from an infinite capacity solution that is not resource consistent, the ISES^{iC} performs iterative sampling for a while then pick the best fixed-time schedule, according to the makespan value, and call the robustify (sketched as a circle whose size is related to the number of performed iterations — see later). The GRASP^{iC} follows the different pattern of simply looking for different start schedule using the randomized ESTA then calling the robustification.

j30	**|flex|**	**|fldt|**	**cpu**	**npc**	**mk**
ISES^{iC}+MINID$_{flex}$	0.472	0.603	6.63	30.20	98.52
ISES^{iC}+MINID$_{flex}$ +MCS linear	0.472	0.607	8.22	30.26	98.12
ISES^{iC}+MINID$_{flex}$ +MCS quadratic	0.475	0.601	9.44	30.09	98.26
ISES^{iC}+MINID$_{fldt}$	0.451	0.627	6.63	30.64	98.44
ISES^{iC}+MINID$_{fldt}$ +MCS linear	0.446	0.631	8.22	30.64	98.10
ISES^{iC}+MINID$_{fldt}$ +MCS quadratic	0.448	0.626	9.42	30.74	98.22

Table 1: ISES^{iC}

j30	**|flex|**	**|fldt|**	**cpu**	**npc**	**mk**
GRASP^{iC}+MINID$_{flex}$	0.461	0.668	6.43	29.17	105.25
GRASP^{iC}+MINID$_{flex}$ +MCS linear	0.473	0.672	7.49	28.54	104.65
GRASP^{iC}+MINID$_{flex}$ +MCS quadratic	0.475	0.670	8.54	28.85	104.86
GRASP^{iC}+MINID$_{fldt}$	0.443	0.686	6.43	29.36	105.08
GRASP^{iC}+MINID$_{fldt}$ +MCS linear	0.448	0.687	7.49	28.85	104.56
GRASP^{iC}+MINID$_{fldt}$ +MCS quadratic	0.449	0.689	8.53	28.99	104.59

Table 2: GRASP^{iC}

Experimental Evaluation

We apply the two approaches to two well-known RCPSP/max benchmark sets, j30 and j100. The benchmark set j30 is composed of 270 instances each of them with 30 activities and 5 resources, while the j100 is composed of 540 instances each of them with 100 activities and 4 resources.

Regarding ISES^{iC}, at the first step the ISES algorithm is used considering 10 restarts as termination criterion. After that, a $\mathcal{POS}$ is searched for applying iterative chaining with a number of iterations equals to 100. The implementation of the GRASP^{iC} method is based on 10 iterations of the main loop, where an initial fixed-time schedule is computed at each step, and on 10 iterations in iterative improvement applied to robustify, where a different $\mathcal{POS}$ is computed at each step. Thus in both the approaches the best $\mathcal{POS}$ is selected among a set of 100 alternatives. We recall that in both cases the iterative chaining method is used with the MINID heuristic. In order to have a fair evaluation of the two criteria, we normalize the results according to an upper bound. The latter is obtained for each metric $\mu()$ considering the value $\mu(\mathcal{P})$ that is the quality of the network that represents the initial temporal structure of the problem.

Therefore, in the tables below, for each algorithm we report the following values: the normalized values, $|flex|$ and $|fldt|$, of the two robustness metrics, the CPU time requested (in seconds), the number of posted constraints, npc, and the makespan, mk.

The set j30. Table 1 contains the results obtained using the six versions of the ISES^{iC} method[5]. An analysis of the result presented shows us as the values of both the metrics do not show great difference among the different algorithm variants. In fact the values of $|flex|$ range from 0.472 to 0.475 and the results of $|fldt|$ from 0.626 to 0.631. Regarding the two metrics introduce above, it is possible to notice a twofold behavior. In fact the values of $flex$ are close or equals to the best value, 0.475, obtained in (Policella *et al.* 2004a)[6]. This is because in both the procedure the iterative chaining method with MINID has an important rule in increasing the flexibility (in terms of $flex$) of the final partial order schedule[7]. On the contrary, the same behavior is not confirmed by the results obtained through the ISES^{iC} variants which try to optimize the $fldt$ metric. In this case we have a decreasing of the quality: from 0.670 to 0.631. A different behavior is obtained in case of the GRASP^{iC} variants (see Table 2). Both considering $flex$ and $fldt$, the methods are able to achieve good quality solutions. In fact in the case of $flex$, like in the case of ISES^{iC}, we achieved results that are close or equals to the current best (0.475). It is worth noting that in the case of $fldt$ the same results obtained in (Policella *et al.* 2004a) are achieved. This confirm the relevance of the improvements in the chaining algorithms.

The set j100. Table 3 and Table 4 show the results obtained in the case of the benchmark j100. In this case the difference of the two problem are smoother than in the previous benchmark. In case of $flex$ we have that the ISES^{iC} variants are better of the GRASP^{iC} (but with an improvement of only 2%). The opposite result is achieved in case of $fldt$: in this case we have 0.642 and 0.634 respectively for the best GRASP^{iC} variant and the best ISES^{iC} variant (the percentage difference is about 1%). The difference behavior observed in

[5]These are obtained by the combination of the three heuristic with the two robustness criteria used during the iterative chaining process.

[6]This result is obtained by using ESTA^{iC}+MINID$_{flex}$ +MCS quadratic with 100 iterations.

[7]Of course there is a noticeable difference in terms of CPU time, where we have respectively 9.44 seconds for the ISES^{iC}+MINID$_{flex}$ +MCS quadratic version and 1.21 for the ESTA^{iC}+MINID.

j100	|flex|	|fldt|	cpu	npc	mk
ISES^{iC}+MINID$_{flex}$	0.211	0.620	44.80	42.57	414.73
ISES^{iC}+MINID$_{flex}$ +MCS linear	0.211	0.619	43.63	42.22	413.67
ISES^{iC}+MINID$_{flex}$ +MCS quadratic	0.211	0.617	46.05	42.20	413.69
ISES^{iC}+MINID$_{fldt}$	0.205	0.634	44.85	42.37	414.61
ISES^{iC}+MINID$_{fldt}$ +MCS linear	0.205	0.633	43.64	42.11	413.40
ISES^{iC}+MINID$_{fldt}$ +MCS quadratic	0.205	0.632	46.00	42.17	413.47

Table 3: ISES^{iC}

j100	|flex|	|fldt|	cpu	npc	mk
GRASP^{iC}+MINID$_{flex}$	0.207	0.630	26.67	38.80	437.77
GRASP^{iC}+MINID$_{flex}$ +MCS linear	0.207	0.627	27.01	38.94	437.20
GRASP^{iC}+MINID$_{flex}$ +MCS quadratic	0.208	0.629	27.85	38.84	437.46
GRASP^{iC}+MINID$_{fldt}$	0.201	0.641	26.74	39.11	436.32
GRASP^{iC}+MINID$_{fldt}$ +MCS linear	0.201	0.641	27.01	38.79	435.72
GRASP^{iC}+MINID$_{fldt}$ +MCS quadratic	0.202	0.642	27.85	38.77	435.79

Table 4: GRASP^{iC}

case of j100 with respect to the benchmark j30 underscores a distinction 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 in the resource usage.

Discussion: makespan versus robustness

The tradeoff between makespan and robustness highlighted above is worth a comment. We have seen before the results of the ISES^{iC} method which is based on the use of ISES to optimize the makespan of the initial schedule as well as to increase the efficiency of the solving process. Those results has showed that the optimization of the makespan value reduces the ability to obtain robust schedules especially if the fluidity metric is taken into account.

Figure 5 underlines possible motivations which can lead to different behaviors between flexibility and fluidity with respect to the makespan optimization factor. Fig. 5(a) presents a possible problem. This is a one resource problem in which three activities have to be scheduled. These are ordered according to the constraints in the figure, i.e., between a and b there is a simple precedence constraint[8] while between a and c the constraint specifies a time window $[1, 3]$, i.e., c cannot start more than 3 time-units after or 1 time-unit before the end of the activity a. Furthermore, the size of each activity describes both its duration (the width) and its resource need (the height). Additionally both a and b require one resource unit for two time units while c has a duration equals to 3 and a resource requirement of 2. To complete the description of the problem, the resource capacity is equals to 2.

Figure 5(b) and 5(c) show two different fixed-time schedules with the associated partial order schedule (the darker arrows represent the additional constraints necessary to obtain a flexible solution). Note that in this case for each fixed-time solution there is a unique partial order schedule: in fact any of the two proposed solutions gives a complete linearization of the activities. The two schedules have different makespan, respectively 7 and 8.

Let us now consider first the flexibility metric. If we look at the two partial order schedules (on the right hand side of Fig. 5(b) and 5(c)) it is possible to notice that in both cases there is the same $flex$ value. In fact in both cases the $flex$ value is zero because there is no pair of un-ordered activities. Now we can shift the attention toward the fluidity metric. This metric considers the slack value between any pair of activities, that is, the minimum and maximum distance between them. In the figure, for the pair (a, c) we have the same value for both solutions, $dist(a, c) = [1, 3]$, which stems from the time window constraint defined between the two activities. On the other pair instead we have two different values: $dist(a, b) = [0, 1]$ in the case in Fig. 5(b) and $dist(a, b) = [4, H - 4]$ in the case in Fig. 5(c). This clearly shows that the flexibility value for the sub-optimal solution is greater than the one in Fig. 5(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. 5(b) while for the case in Fig. 5(c) $dist(c, b) = [0, H - 8]$. The problem is that in the schedule with the optimum makespan (Fig. 5(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, no chaining method can overcome this problem.

It is possible to note that this is peculiar characteristic of the RCPSP/max problems and in particular this is due to the presence of maximum distance constraints. In fact if the maximum constraint between a and c did not exist, activity b would have the ability to move back and forth in a larger interval thus yielding in a more flexible solution.

[8]Where H represents the temporal horizon of the problem.

(a) ISES^{iC}

(b) GRASP^{iC}

Figure 3: Current contribution

(a) ISES^{iC}

(b) GRASP^{iC}

Figure 4: ISES^{iC} vs. GRASP^{iC}

(a) Initial problem

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

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

Figure 5: Dependency between makespan and robustness

Further remarks. The use of more optimized initial schedule biases the robustify phase against the construction of flexible partial order schedule. In fact the tightness (makespan) of the initial solution can preclude the achievement of good solutions. This behavior is justified from the different nature of the two metrics: $flex$ is a qualitative criterion whereas the $fldt$ is a more quantitative metric. Therefore a schedule with a better makespan value presents an allocation of the activities more compact giving fewer degree of intervention to the iterative chaining procedure. Finally while in (Policella *et al.* 2004a) has been proved that the robustify step does not deteriorate the results obtained in the solve phase, we can not claim the same for the effects of some characteristics of the fixed-time solutions on the final flexible schedule.

Conclusions

This work has considered the Solve-and-Robustify approach introduced in (Policella *et al.* 2004b; 2004a) and has complemented previous results by analyzing some directions to discover the influence of different fixed-time schedule on the final partial order schedule. In particular, two more sophisticated approaches based on Solve-and-Robustify have been defined and evaluated. The former is based on the idea of developing flexible solutions from makespan optimized solutions. The second approach instead has been obtained following the GRASP paradigm, where the robustify step can be considered as a local search procedure.

The results of the experimental evaluation have shown an interesting trade-off between the makespan and the more quantitative of the robustness metrics used, $fldt$. In practice the use of more optimized initial schedule biases the robustify phase against the construction of flexible partial order schedule. In fact the tightness (makespan) of the initial solution can preclude the achievement of good flexible solutions. Therefore while the robustify step does not deteriorate the results obtained in the solve phase some characteristics of the fixed-time solutions may lower the robustness of the final partial order schedules.

Acknowledgments

Amedeo Cesta, Angelo Oddi, and Nicola Policella's work is partially supported by MIUR (Italian Ministry for Education and Research) under project RoboCare. 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 contract F30602-00-2-0503, by the National Science Foundation under contract # 9900298 and by the CMU Robotics Institute.

References

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

Artigues, C., and Roubellat, F. 2000. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research* 127(2):297–316.

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*, 214–223.

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.

Cicirello, V. A., and Smith, S. F. 2004. Heuristic selection for stochastic search optimization: Modeling solution quality by extreme value theory. In *Proceedings of CP. Lecture Notes in Computer Science N.3258*, 197–211.

Dorndorf, U.; Pesch, E.; and Phan-Huy, T. 2000. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science* 46:1365–1384.

Leus, R., and Herroelen, W. 2004. Stability and Resource Allocation in Project Planning. *IIE Transactions* 36(7):667–682.

Oddi, A., and Smith, S. F. 1997. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14*th *National Conference on Artificial Intelligence (AAAI-97)*, 308–314.

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

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

Policella, N. 2005. *Scheduling with Uncertainty: a Proactive Approach using Partial Order Schedules*. Ph.D. Dissertation, University of Rome "La Sapienza".

Resende, M., and Ribeiro, C. 2002. Greedy Randomized Adaptive Search Procedures. In Glover, F., and Kochenberger, G., eds., *Handbook of Metaheuristics*. Kluwer Academic Publishers. 219–249.

Smith, T. B., and Pyle, J. M. 2004. An effective algorithm for project scheduling with arbitrary temporal constraints. In *Proceedings of AAAI*, 544–549.