Generating Robust Partial Order Schedules

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

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

Abstract. This paper considers the problem of transforming a resource feasible, fixed-times schedule into a *partial order schedule* (POS) to enhance its robustness and stability properties. Whereas a fixed-times schedule is brittle in the face of unpredictable execution dynamics and can quickly become invalidated, a POS retains temporal flexibility whenever problem constraints allow it and can often absorb unexpected deviation from predictive assumptions. We focus specifically on procedures for generating *Chaining Form* POSs, wherein activities competing for the same resources are linked into precedence chains. One interesting property of a Chaining Form POS is that it is "makespan preserving" with respect to its originating fixed-times schedule. Thus, issues of maximizing schedule quality and maximizing schedule robustness can be addressed sequentially in a two-step scheduling procedure. Using this approach, a simple chaining algorithm was recently shown to provide an effective basis for transforming good quality solutions into POSs with good robustness properties. Here, we investigate the possibility of producing POSs with better robustness and stability properties through more extended search in the space of Chaining Form POSs. We define two heuristics which make use of a structural property of chaining form POSs to bias chaining decisions. Experimental results on a resource-constrained project scheduling benchmark confirm the effectiveness of our approach.

1 Introduction

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 unexpected events quickly invalidate the schedule's predictive assumptions and bring into question the continuing validity of the schedule's prescribed actions. The lifetime of a schedule tends to be very short, and hence its optimizing advantages are generally not realized.

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

Part of the schedule brittleness problem stems from reliance on a classical, fixed-times formulation of the scheduling problem, which designates the start and end times of activities as decision variables and requires specific assignments to verify resource feasibility. By instead 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 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.

One important open question, though, is how to generate flexible schedules with good robustness properties. In [1] a two-stage approach to generating a flexible schedule is introduced as one possibility. Under this scheme, a feasible fixed-times schedule is first generated in stage one (in this case, an early start times solution), and then, in the second stage, a procedure referred to as *chaining* is applied to transform this fixed-times schedule into a temporally flexible schedule in so-called *Chaining Form*. Concepts similar to the idea of a *Chaining Form* schedule have also been used elsewhere: for example, the Transportation Network introduced in [2], and the Resource Flow Network described in [3] are based on equivalent structural assumptions. The common thread underlying these particular representations 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".

In a recent paper [4], this approach – find a solution then make it flexible – was shown to produce schedules with better robustness properties than a more direct, least-commitment generation procedure. These results establish the basic viability of a chaining approach. At the same time, the procedure used in this work to produce a Chaining Form solution was developed simply to provide a means of transforming a given fixed-times schedule into a temporally flexible one. Although final solutions were evaluated with respect to various robustness properties, no attention was given to the potential influence of the chaining procedure itself on the properties exhibited by the final solution. In this paper, we examine the problem of generating a schedule in Chaining Form from the broader perspective of producing temporally flexible schedules with good robustness properties, and investigate the design of informed chaining procedures that exploit knowledge of these properties to increase the robustness of the final generated solution. We first establish basic properties that indicate the potential of extended search in the space of chaining solutions, and also show that a Chaining Form schedule is "makespan preserving" with respect to its originating fixed-times schedule. Then we define two heuristics explicitly designed to take advantage of *Chaining Form* analysis and to search for solutions with good robustness properties. Experimental results on

resource-constrained project scheduling benchmark problems confirm the effectiveness of these search procedures. We begin by establishing a reference scheduling problem and summarizing the basic notion of schedule robustness that underlies our work.

2 Scheduling Problem

We adopt the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max, as a reference problem [5]. The basic entities of interest in this problem are *activities*. The set of activities is denoted by $V = \{a_1, a_2, \ldots a_n\}$. Each activity has a fixed *processing time*, or *duration*, d_i. Any given activity must be scheduled without preemption.

A *schedule* is an assignment of start times to activities $a_1, a_2, \ldots a_n$, i.e. a vector $S = (s_1, s_2, \ldots, s_n)$ 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:

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

Schedules are subject to 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,

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

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 $S = (s_1, s_2, \ldots, s_n)$ 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 of 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.

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

A schedule S is called *feasible* if it is both time and resource feasible.

3 Robustness & Flexible Schedules

As indicated above, we are concerned with the generation of schedules that offer some degree of robustness in the face of a dynamic and uncertain execution environment. In any given scheduling domain, there can be different sources of executional uncertainty:

durations may not be exactly known, there may be less resource capacity than expected (e.g., due to machine breakdowns), or new tasks may need to be taken into account.

The concept of robustness has been approached from different perspectives in previous work. Some definitions of robustness have emphasized the ability to preserve some level of solution quality, such as preservation of makespan in [6, 3]. Alternatively, other work has considered robustness to be an execution-oriented quality. For example, in [7] robustness is defined as a property that is dependent on the repair action entailed by a given unexpected event. This view singles out two distinct, co-related aspects of robustness: the ability to keep pace with the execution (implying bounded computational cost) and the ability to keep the *evolving* solution stable (minimizing disruption). In fact a small perturbation to a scheduled event can, in general, cause a large ripple of changes through the current schedule.

Our view of robustness is also execution-oriented. We consider a solution to a scheduling problem to be *robust* if it provides two general features: (1) the ability to absorb external events without loss of consistency, and (2) the ability to keep the pace with execution. Our approach is to focus on generating 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 such a schedule is indicative of its robustness. More precisely, our approach (see also [4]) adopts a graph formulation of the scheduling problem and focuses on generation of *Partial Order Schedules* (POSs). Within a POS, each activity retains a set of feasible start times, and these options provide a basis for responding to unexpected disruptions. An attractive property of a POS is that reactive response to many external changes can be accomplished via simple propagation in an underlying temporal network (a polynomial time calculation); only when an external change exhausts all options for an activity it is necessary to recompute a new schedule from scratch. Given this property and given a predefined horizon H, the *size* of a POS – the number of fixed-times schedules (or possible execution futures) that it "contains" – is suggestive of its overall robustness[3]. In general, the greater the size of a POS the more robust it is. Thus, our challenge is to generate POSs of maximum possible size. Before considering this challenge we first define the notion of a Partial Order Schedule (POS) more precisely.

3.1 Partial Order Schedules

We represent a scheduling problem P as the 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. In particular for each constraint of the form $l_{ij}^{min} \leq s_j - s_i \leq l_{ij}^{max}$, there is an edge $(a_i, a_j) \in E_P$ with label $[l_{ij}^{min}, l_{ij}^{max}]$.

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. In particular, let $F \subseteq V$ be any subset of activities such that there exists a time t where $\sum_{s_i \leq t < e_i} req_{ik} > c_k$. This subset is called a forbidden set [5] (or contention peak), and a *minimal forbidden set* (or resource conflict or *minimal*

[3] The use of an horizon is justified by the need to compare POSs of finite size.

critical set) is a set $F_{min} \subseteq F$ such that each of its proper subsets is not a forbidden set. Any minimal forbidden set F_{min} is removed by adding a single precedence constraint between any pair of activities in F_{min}, and these additional constraints become the elements of E_R. Noting these concepts and recalling that 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:

> Given a scheduling problem, $G_P(V_P, E_P)$, a *Partial Order Schedule* is a graph $POS(V_P, E_P \cup E_R)$ such that any *time feasible* schedule is also a *feasible* schedule.

Before concluding, we introduce two further concepts which will be used in the remaining of the paper: the *earliest start schedule* of a POS, $ES(POS)$, 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, $s_i = est(a_i)$ for $1 \leq i \leq n$. Finally, the makespan of a POS is defined as the makespan of its earliest start schedule, that is, $mk(POS) = max_{a_i \in V}\{est(a_i) + d_i\}$.

4 Partial Order Schedules in Chaining Form: Basic Properties

In our previous work [1] we developed a two-stage procedure for generating a POS, based on generation and subsequent transformation of a "fixed-times" schedule. In this procedure, the second transformation step is accomplished by a *chaining* procedure, so called because fixed-times commitments are converted into sequences (chains) of activities to be executed by various resources. In [4], we showed this approach to be capable of generating POSs more efficiently than a least commitment POS generation procedure while simultaneously producing POSs with better robustness properties. These results indicate the potential of this two-stage approach for generating robust schedules. At the same time, the chaining procedure underlying this work was developed originally to provide a means for efficiently generating POSs.

Our goal in this section is to examine the concept of Chaining Form solutions from the broader perspective of generating robust POSs and to establish properties that can guide the development of chaining procedures capable of generating more robust POSs. We describe a canonical graph form, the Chaining Form POS^{ch}, for representing a POS and show that any given POS is expressible in this form. Thanks to this result we can restrict our attention to the design of procedures that explore the space of partial order schedules in chaining form, POS^{ch}. This will then be accomplished introducing a family of operators for transforming a generic fixed-times schedule into a partial order schedule in chaining form, POS^{ch}.

As introduced in [1], the concept of chaining form refers to a POS in which a *chain* of activities is associated with each unit of each resource. In the case of scheduling problems involving activities which require only a single unit of a resource, a solution is in a *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 $a_{j,i-1}$ will be executed before $a_{j,i}$, $a_{j,i-1} \prec a_{j,i}$ for $i = 1, \ldots, N_j$. This definition can be easily extended to the general case where each activity can require one or more units of one or more resources.

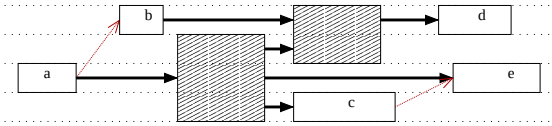


Fig. 1. A partial order schedule in chaining form

In such a case, any activity requiring $req_{ik} > 1$ resource units can be replaced with a set of req_{ik} activities (each requiring one unit) that are constrained to execute in parallel. As a consequence, in the general case, an activity will be allocated to as many chains as necessary to fulfill its resource requirements.

Figure 1 represents a partial order schedule in chaining form for a problem with a single resource r_k with capacity $c_k = 4$. The bold arcs represent the set of chains and the thin arcs designate further constraints defined in the problem. The size of each activity reflects both its duration and its resource requirement, respectively, the length represent the duration while the height the request. Hence, the gray activities will require more than one unit of resource. This implies that both of them will be allocated to more than one chain.

By definition, a solution in chaining form is a partial order schedule. It is also possible to prove that any partial order schedule POS admits at least an equivalent POS in chaining form[4].

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

Proof. Let $\overline{POS}(V_P, \overline{E})$ be the transitive closure of the graph 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 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 $POS^{ch}(V_P, E_P \cup E^{ch})$ with $E^{ch} \subseteq \overline{E}$ such that POS^{ch} represents at least the same set of solutions of POS. In fact, given the set $\overline{E}$, for each resource r_k, we can always select a subset of simple precedence constraints $E_k^{ch} \subseteq \overline{E}$ 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 $(a_{j,i-1}, a_{j,i}) \in E_k^{ch} \subseteq \overline{E}$ with $i = 1, \ldots, n_j$ and $E^{ch} = \bigcup_{k=1}^{m} E_k^{ch}$.

Proof by contradiction: let us assume as not possible the construction of such a POS^{ch}. Then, there is at least one resource r_k for which there is an activity a_k which 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 POS is a partial order schedule. Thus, it is always possible to build a POS^{ch} from a POS with $E^{ch} \subseteq \overline{E}$. □

Given this result, we can restrict our attention, without loss of generality, to the set of POSs which have a chaining form. Hence, a general operator for transforming a fixed-times schedule into a POS can be defined as follows:

[4] An analogous result is proved in [3].

Chaining(P, S)
Input: A problem P and one of its fixed-times schedules S
Output: A partial order solution POS^{ch}
1. $POS^{ch} \leftarrow P$
2. Sort all the activities according to their start times in S
3. Initialize the all chains empty
4. **for each** resource r_j
5. **for each** activity a_i
6. **for** 1 **to** req_{ij}
7. $k \leftarrow SelectChain(a_i, r_j)$
8. $a_k \leftarrow last(k)$
9. $AddConstraint(POS^{ch}, a_k \prec a_i)$
10. $last(k) \leftarrow a_i$
11. **return** POS^{ch}

Fig. 2. Basic Chaining procedure

Definition 1 (Chaining operator). *Given a fixed-times schedule S a chaining operator $ch()$ is an operator that applied to S returns a partial order schedule*

$$POS_S^{ch} = ch(S)$$

such that POS_S^{ch} is in chaining form and S is contained in the set of solution it describes.

Figure 2 describes a basic chaining operator. The first step sorts all activities according to their start times in the schedule S. Then the activities are incrementally allocated on the different chains. We note that in case where an activity requires more than one unit of one or more resources, it will be allocated to a number of chains equal to the overall number of resource units it needs. The function $SelectChain(a_i, r_j)$ is the core of the procedure; it can admit different definitions giving different results. A basic implementation chooses, for each activity, the first available chain of r_j. Given an activity a_i, a chain k is *available* if the end time of the last activity allocated on it, $last(k)$, is not greater than the start time of a_i. Note that since the input to a chaining operator is a consistent solution it will always be possible to find the chains that the activity a_i needs.

A chaining operator can be seen as a post-processing step which dispatches (or allocates) tasks to specific resource units once that a resource feasible (fixed-times) solution has been built. Given that a common objective of the first step in many scheduling domains will be to construct a feasible fixed-times solution that minimizes makespan, the following property plays an important role:

Property 1. Given a fixed-times schedule S and its POS_s^{ch}

$$mk(ES(POS_s^{ch})) \leq mk(S).$$

That is, the makespan of the earliest solution of POS_S^{ch} is not greater than the makespan of the input solution S.

By definition S is one of the solutions represented by POS_S^{ch} then $mk(ES(POS_S^{ch})) \leq mk(S)$. 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.

5 Generating More Robust Schedules via Iterative Sampling

The chaining operator introduced in the previous section transforms a feasible fixed-times solution into a POS in chaining form by dispatching activites to specific resource units[5]. In the basic implementation shown in Fig. 2 this dispatching process is carried out in a specific deterministic manner; the $SelectChain(a_i, r_j)$ sub-procedure always dispatches the next activity a_i to the first available resource unit (chain) associated with its required resource r_j. However, since there are generally choices as to how to dispatch activities to resource units, it is possible to generate different POSs from a given initial fixed-times schedule, and these different POSs 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 of the final POS produced by the chaining process. Given the results of the previous section, we can search this space of possible POS^{ch} 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 POSs in chaining form. Specifically, the chaining operator described in Fig. 2 is executed n times starting from the same initial fixed-times solution, and non-determinism is added to the strategy used by $SelectChain(a_i, r_j)$ to obtain different POSs across iterations. Each POS generated is evaluated with respect to some designated measure of robustness, and the best POS found overall is returned at the end of the search. In Section 6.1 we describe the two metrics used in the actual implementation to approximate POS size, and explain why it is not possible to directly analyze the size of a POS.

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 POS^{ch}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 a more informed heuristic for dispatching activities to chains, it is useful to examine the structure of solutions produced by the chaining procedure. Consider the example in Fig. 1. Note that both 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 POS's ability to both absorb the event and retain flexibility for

[5] Note that such a procedure is required to enable schedule execution.

future changes. Hence it is desirable to minimize the number of synchronization points where possible.

A synchronization point can originate from one of two different sources:

- a constraint defined in the problem which relate 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 synchronzation point is strictly a consequence of the problem. However, in the second case, the synchronization point could follow from the way that the chains are built and might be preventable. For example, consider the POS given in Fig. 3. 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 POS in Fig. 1 the two gray activities span all four chains. They effectively split the solution into two parts, and the whole 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.

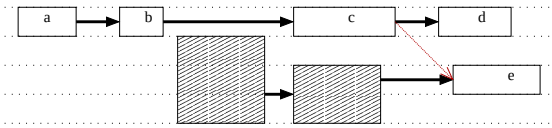


Fig. 3. A more flexible POS

Based on this observation, we define a first heuristic chain selection procedure that favors allocation of activities to common chains. 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 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 POS in Fig. 3.

The example in Figure 3 allows us to show a second anomaly that can be observed in chaining form POSs. 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

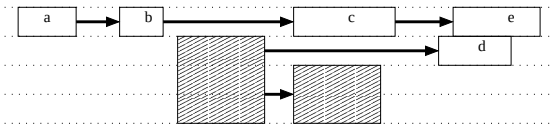


Fig. 4. A even more flexible POS

selection procedure, which augments the first by replacing the random selection of the first chain for a given activity (step (**1**)) with a more informed choice that takes into account existing ordering relations with those activities already allocated in the chaining process. More precisely, step (**1**) of our first heuristic is replaced by the the following sequence of steps: (**1a**) the chains k for which their last element, $last(k)$, is already ordered wrt activity a_i, are collected in the set P_{a_i}. Then (**1b**) if $P_{a_i} \neq \emptyset$ a chain $k \in P_{a_i}$ is randomly picked, otherwise (**1c**) a chain k is randomly selected among the available ones. (**1d**) 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 4 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. 3 is eliminated.

6 Experimental Evaluation

In this section we evaluate the performance of the algorithms proposed in Section 5 with respect to a set of metrics to evaluate both solution's robustness and stability. Our comparison is based on the benchmark $J30$ defined in [8], which consists of 270 problem instances with 30 activities and 5 resources. The remainder of this section is organized as follows. We first present two metrics which characterize robustness by approximating the size of a POS and discuss experimental results obtained relative to these metrics with various proposed algorithms. Next, we introduce a complementary metric that characterizes solution *stability* and additionally evaluate our experimental results with respect to this metric.

6.1 Measuring Robustness

As suggested earlier, a POS represents a set of temporal solutions that are also resource feasible, and this set provides a means for tolerating some amount of executional 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 POS. *Hence, it follows that within the same horizon H, the greater the number*

of solutions represented in a POS, the greater its robustness. It is worth noting that in order to have a finite number of solutions, we always assume that all activities in a given problem must be completed within a specified finite horizon. In particular, we consider a default horizon H imposed on each problem equal to the sum of all activity durations d_i and the sum of all the minimal time legs l_{ij}^{min}. Unfortunately counting the number of solutions in a POS is a *#P-complete* problem (e.g., see [9] page 330). For this reason, in the following section we will use two measures which are indirectly related to the number of solutions in a POS.

The first metric is taken from [1] and is defined as the average width, relative to the temporal horizon, of the temporal slack associated with each pair of activities (a_h, a_l):

$$fldt = \sum_{h \neq l} \frac{Slack(a_h, a_l)}{H \times n \times (n-1)} \times 100 \quad (4)$$

where H is the horizon of the problem, n is the number of activities and $Slack(a_h, a_l)$ is the width of the allowed distance interval between the end time of activity a_h and the start time of activity a_l. This metric characterizes the *fluidity* of a solution, i.e., the ability to use flexibility to absorb temporal variation in the execution of activities. The higher the value of $fldt$, the less the risk of a "domino effect", i.e. the higher the probability of localized changes.

A second measure is taken from [10] and is called $flex$. This measure counts the *number of pairs of activities in the solution which are not reciprocally related by simple precedence constraints*. This metric provides an analysis of the configuration of the solution. The rationale for this measure is that when two activities are not related it is possible to move one without moving the other one. Hence, the higher the value of $flex$ the lower the degree of interaction among the activities.

6.2 Results

Table 1 summarizes the main results[6], in particular we compare the following three sets of chaining methods:

- the basic chaining operator as described in Figure 2, named CHN;
- the iterative sampling procedure which maximizes only the flexibility metric, $flex$. There are three different variants: the pure randomized version, IS_{flex}, the first heuristic biased version aimed at maximizing chain overlap between activities that require multiple resource units, ISH_{flex}, and the enhanced heuristic biased version which adds consideration of extant activity ordering constraints, ISH^2_{flex}.
- same as above with the difference that the optimized parameter is the fluidity, $fldt$. In this case the procedures are named IS_{fldt}, ISH_{fldt} and ISH^2_{fldt}.

The results shown in Table 1 are the average values obtained over the subset of solved problems in the $J30$ benchmark. For each procedure five parameters value 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 CPU time,

[6] All algorithms presented in the paper are implemented in C++ on a Pentium 4-1,500 MHz processor under Linux OS.

	$flex$	$fldt$	cpu	npc	mk
CHN	7.0	27.4	5.4	38.7	107.1
IS_{flex}	7.7	28.8	82.7	44.8	106.8
ISH_{flex}	9.1	29.2	82.4	39.1	106.7
ISH^2_{fldt}	13.3	31.3	79.2	28.3	105.7
IS_{fldt}	7.3	29.3	74.8	44.7	106.1
ISH_{fldt}	8.5	30.3	72.0	39.6	106.4
ISH^2_{fldt}	12.8	32.3	69.1	28.9	105.4

Table 1. Performance of the algorithms

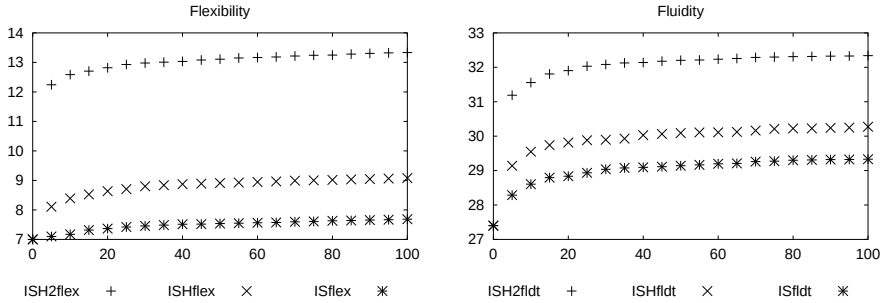
we include both the time to find an initial fixed-times solution and the time required by the chaining procedure. In the case of iterative procedures, the values shown reflect 100 iterations.

Analyzing the results, we first observe that all search procedures outperform the basic chaining procedure, it is clearly worthwhile to explore the space of possible POS^{ch} derivable from a given fixed-times solution S if the goal is to maximize solution robustness. In fact, all search strategies are also seen to produce some amount of improvement in solution makespan, an interesting side benefit. We further observe that the two heuristic strategies based on minimizing the number of synchronization points clearly outperform the basic iterative randomized procedure. The iterative sampling procedure with heuristic bias, ISH_{flex}, is able to improve 30% over the basic chaining results while the version using the enhanced heuristic, ISH^2_{flex}, obtains a gain of about 90% (from 7.0 to 13.3). The informed selection of the first chain thus is clearly a determining factor in achieving good quality solutions. These results are also confirmed by the corresponding procedures for the $fldt$ parameter. In this case, improvement ranges from about 10% for the first heuristic ISH_{fldt}, to about 18%, for ISH^2_{fldt}.

Another discriminating aspect of the performance of the enhanced heuristic ISH^2_{fldt} is its ability to take advantage of pre-existing precedence constraints and reduce the number of posted constraints[7] (see npc column in Table 1). 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 lower number of constraints may contribute to compression of the critical path. On the other side of the coin, use of the iterative procedure incurs a non negligible additional computational cost.

Figure 5 highlights a further aspect which differentiates the heuristic biased iterative procedures from the pure randomized procedure. This picture plots the value of the best solution found by each iterative procedure as the search progresses (with respect to the number of iterations). Fig. 5(a) represents the results obtained when the metric $flex$ is taken into account while in Fig. 5(b) the procedures aim at optimizing the $fldt$ value. The heuristic biased procedures are seen to find better solutions at a much faster rate than the basic randomized procedure, and quickly reach solutions better than the best solutions generated by the basic randomized procedure (as shown in Table 1). For instance the best solution obtained by ISH_{flex} after 10 iterations is higher quality than

[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.



(a) Flexibility vs number of iterations (b) Fluidity vs number of iterations

Fig. 5. Iterative sampling's efficiency

the solution obtained by IS_{flex} after 100 iterations (see Fig. 5(a)); likewise, ISH^2_{flex} (ISH^2_{fldt}) are able to obtain better solutions that can be obtained by any other procedures in just a few iterations. It is clear that the use of heuristic bias focuses the search on a more significant region of the search space for both robustness metrics, and that this bias both accelerates and enhances generation of better solutions.

6.3 Evaluating Schedule Stability

As introduced in the first part of the work, it is possible to take a different point of view in maximizing the robustness of a POS: we can search for a *stable set of solutions*. That is, we search for a POS that is both (1) capable of finding new start time assignments for all activities consistent with the new constraints imposed by an exogenous event (exhibiting robustness), and (2) capable of *absorbing* the modification (minimizing its broader impact). For example, suppose that the start time s_i of the activity a_i is delayed by Δ_{in}. Then we would like the average delay of other activities Δ_{out} to be much much smaller: $\Delta_{out} \ll \Delta_{in}$. To evaluate the stability of a solution we consider a single type of modification event: the start time s_i of a single activity a_i with *window* of possible start times $[est(a_i), lst(a_i)]$ is increased to a value $s_i + \alpha w_i/100$, where $0 \leq \alpha \leq 100$ and $w_i = lst(a_i) - est(a_i)$. In the following we give a definition of *stability* and re-evaluate the same results obtained in the previous section with respect to this metric. Our goal is to understand the correlations with other proposed metrics.

A more operative definition of stability is given by the following formula:

$$stby(\alpha) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{\delta_j(\alpha)}{w_i} \tag{5}$$

where the stability $stby(\alpha)$ is defined as the average value $\frac{\delta_j(\alpha)}{w_i}$ over all pairs (a_i, a_j), with $a_i \neq a_j$, when an increase of start time $\alpha w_i/100$ is performed on each activity start time separately. We observe that the single value $\frac{\delta_j(\alpha)}{w_i}$ 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 $s_i + \alpha w_i/100$. Note that, by definition, the $stby(\alpha)$ value is included in the interval $[0, 1]$.

α	CHN	ISH_{fldt}	ISH^2_{fldt}	ISH_{flex}	ISH^2_{flex}	LB
1	0.09	0.08	0.07	0.08	0.07	0.05
2	0.29	0.27	0.24	0.27	0.25	0.15
4	0.89	0.82	0.76	0.82	0.77	0.48
8	2.51	2.34	2.22	2.33	2.21	1.48
16	6.59	6.26	6.01	6.14	5.93	4.33
25	11.78	11.28	10.85	11.00	10.68	8.14
50	27.42	26.43	25.37	25.66	24.92	19.84
75	43.67	42.22	40.43	40.91	39.68	32.06
100	**60.18**	58.24	55.69	56.40	**54.65**	44.42

Table 2. Evaluation of the stability (percentage value)

Table 2 compares the values of the function $stby(\alpha)$ for different values of α (*disruption*). We compare five sets of data: the stability of the solutions obtained with the application of the chaining operator, *CHN*, and the stability obtained with the procedures ISH_{fldt}, ISH_{flex}, ISH^2_{fldt}, ISH^2_{flex}. To enable a better evaluation lower bound values are also given. These are calculated using the initial partial order (i.e. the input problem) without any additional resource constraints. As can be seen, the stability of the set of $J30$ solutions improves over the simple chaining results and the best improvement is obtained with the procedure ISH^2_{fldt}: when the disruption α reaches the maximal value ($\alpha = 100$) the value of the stability is reduced from 60% to about 54%. These results indicate that the solutions generated by the iterative procedures, in addition to exhibiting good properties with respect to solution flexibility, also do not exhibit any negative side-effects with respect to solution stability.

7 Conclusion and Future Work

In this paper, we have considered the problem of transforming a resource feasible, fixed-times schedule into a *Partial Order Schedule* (POS) to enhance its robustness and stability properties. Unlike other related work in this area [2, 3], we consider a complex problem, RCPSP/max, that is not polynomially solvable, and hence enhances the need for flexible solutions.

We focused specifically on the problem of generating POSs in *Chaining Form*, where activities competing for the same resources are linked into precedence chains. The paper pointed out two basic properties: (1) that a given POS can always be represented in Chaining Form; and (2) that chaining - the process of constructing a Chaining Form POS - is makespan preserving with respect to an input schedule. As a consequence, issues of maximizing schedule makespan and maximizing schedule robustness can be addressed sequentially in a two-step scheduling procedure.

On the basis of the first property, we considered the possibility of producing POSs with better robustness and stability properties through more extended search in the space of Chaining Form POSs. In particular, three iterative sampling procedures for chaining are proposed: the first one simply randomizes the choices made by a simple chaining algorithm; the remaining two take account of structural properties of more robust Chaining Form POSs to heuristically bias chaining decisions. To evaluate these procedures, we developed metrics for assessing the robustness and stability of a gen-

erated POS. Experimental results on a set of challenging resource constrained project scheduling benchmarks were shown to confirm the effectiveness of the approach. In general, consideration of Chaining Form POSs emphasizes the presence of synchronization points as obstacles to flexibility, and this fact can be exploited to generate POSs with good robustness properties.

Several directions can be mentioned for future work. One is to study the possibility of further enhancing the flexibility of a Chaining Form solution by dropping the requirement that each chain be totally ordered and instead allow specific subsets of activities allocated to a given chain to remain unordered (and permutable as execution circumstances dictate). The work of [11] provides a starting point for considering this sort of extension. A second possibility for future research is the definition of broader search strategies that use a chaining operator as a core component. In fact, the result obtained by any chaining operator is biased by the initial solution that seeds the chaining procedure. From this perspective, one point to investigate is the relation between the initial solution and the partial order schedule which can be obtained through chaining.

Acknowledgments. Stephen F. Smith's work is supported in part by the Department of Defense Advanced Research Projects Agency and the U.S. Air Force Research Laboratory - Rome, under contracts F30602-00-2-0503 and F30602-02-2-0149, by the National Science Foundation under contract # 9900298 and by the CMU Robotics Institute. Amedeo Cesta, Angelo Oddi, and Nicola Policella's work is partially supported by ASI (Italian Space Agency) under project ARISCOM (Contract I/R/215/02).

We would like to thank an anonymous reviewer for several comments that helped us to improve the presentation

References

1. Cesta, A., Oddi, A., Smith, S.F.: Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In: Proceedings of AIPS-98. (1998)
2. Artigues, C., Roubellat, F.: A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. European Journal of Operational Research **127** (2000) 297–316
3. Leus, R., Herroelen, W.: Stability and Resource Allocation in Project Planning. IIE Transactions **36** (2004) 667–682
4. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Generating Robust Schedules through Temporal Flexibility. In: Proceedings of ICAPS'04. (2004)
5. Bartusch, M., Mohring, R.H., Radermacher, F.J.: Scheduling project networks with resource constraints and time windows. Annals of Operations Research **16** (1988) 201–240
6. Leon, V., Wu, S., Storer, R.: Robustness measures and robust scheduling for job shops. IIE Transactions **26** (1994) 32–43
7. Ginsberg, M.L., Parkes, A.J., Roy, A.: Supermodels and Robustness. In: Proceedings of AAAI-98. (1998)
8. Kolisch, R., Schwindt, C., Sprecher, A.: Benchmark instances for project scheduling problems. In Weglarz, J., ed.: Project Scheduling - Recent Models, Algorithms and Applications. Kluwer, Boston (1998) 197–212
9. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
10. Aloulou, M.A., Portmann, M.C.: An Efficient Proactive Reactive Scheduling Approach to Hedge against Shop Floor Disturbances. In: Proceedings of MISTA 2003. (2003)
11. Artigues, C., Billaut, J., Esswein, C.: Maximization of solution flexibility for robust shop scheduling. European Journal of Operational Research (2004) To appear.