UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XVII CICLO – 2005

Scheduling with Uncertainty
A Proactive Approach using Partial Order Schedules

Nicola Policella

UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XVII CICLO - 2005

Nicola Policella

Scheduling with Uncertainty
A Proactive Approach using Partial Order Schedules

Thesis Committee

Prof. Amedeo Cesta (Advisor)
Prof. Umberto Nanni
Prof. Andrea Schaerf
Prof. Marco Schaerf

Reviewers

Dr. Claude Le Pape
Prof. Kenneth N. McKay
Prof. Stephen F. Smith

AUTHOR'S ADDRESS:
Nicola Policella
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
E-MAIL: policella@dis.uniroma1.it
WWW: http://pst.istc.cnr.it/∼nicola/

To Angelo, Lina, Nicola, and Rosina

Acknowledgment

There are many people whose support and guidance have helped to make this dissertation a reality. I would like to acknowledge as many of them as possible here.

First, I would like to thank my advisor, Amedeo Cesta. None of this would have been possible without his guidance and support. His insights into the research process, his faith in my abilities, and his friendship enabled me to complete this work.

Next, I would like to express my gratitude to the members of the Planning and Scheduling Team for discussions that have lead my thoughts in directions that otherwise may not have been considered fully. I would like to specifically acknowledge: Gabriella Cortellessa, Simone Fratini, Angelo Oddi, Marcelo Oglietti, Federico Pecora, and Riccardo Rasconi.

Thanks to Steve Smith and all the members of the ICL laboratory. They hosted me for more than one year providing me a stimulating environment that has fundamental merits in the outcome of my dissertation.

Thanks also go to the members of my thesis committee, Umberto Nanni, Andrea, and Marco Schaerf, and the additional reviewers, Claude Le Pape and Ken McKay. Their advice and guidance throughout my thesis research have been immeasurable.

Special thanks go to my family for its financial, intellectual and emotional support throughout this long process. Last but not least, I would like to thank my girlfriend, Fausta, for her love, support, and encouragement. I could not have done this without her.

Abstract

Over the last decades, many approaches have been developed for application to a variety of different scheduling problems. Most of these techniques assume erroneously a complete information about the scheduling problem to be solved. Moreover a further limitation arises from the assumption that the schedule will be executed in a well-known environment. Unfortunately, the real word is not so predictable: machines break down, activities last longer than expected and further activities can be inserted into the problem. This creates the need to study specific techniques to face these aspects.

The aim of this dissertation is the production of robust schedules, i.e., solutions that are able to absorb scheduling uncertainty during their execution. In particular we consider the Resource Constrained Project Scheduling Problem with Generalized Precedence Relations (RCPSP/max) as a reference. For this problem we give a definition of schedule robustness in terms of identification of a *Partial Order Schedule* ($\mathcal{POS}$). A $\mathcal{POS}$ is defined as a set of solutions for the scheduling problem that can be compactly represented within a temporal graph. This implicit representation provides both a means to promptly answer to temporal changes (e.g. the durations of the activities change or the start-time of the activities are delayed) and a base to hedge against further changes (e.g., new activities to serve or resource capacity variations).

After defining the $\mathcal{POS}$, we investigate how to generate flexible schedules with good robustness properties. In particular, two main solving methodologies are found. First, a least commitment methodology in which computed bounds on cumulative resource usage are used to identify potential resource conflicts, and progressively winnows the total set of temporally feasible solutions into a smaller set of resource feasible solutions by resolving detected conflicts.

A different, less intuitive, approach to $\mathcal{POS}$ synthesis is based on a goal separation and called *Solve-and-Robustify*: under this schema, a feasible fixed-time schedule is first generated in stage one, and then, in the second stage, the initial solution is transformed into a temporally flexible schedule. In particular we analyze a heuristic technique, called *chaining*, for obtaining $\mathcal{POS}$s starting from a fixed-time schedule obtained with one of the current available techniques.

Contents

Chapter 1

Introduction

This thesis addresses the broad question of how to build schedules that are robust in the face of a dynamic execution environment. This question is of considerable practical importance, as a major obstacle to the use of schedules in practice is their brittleness when activities can not be executed as planned. Different scheduling techniques have been developed to analyze scheduling problems and produce high quality solutions. Unfortunately, in the real world a high degree of uncertainty is present; therefore it is hard to have an exact estimation of the evolution of the world. In the case of a scheduling problem, the latter point may represent a serious restriction to the application of state-of-the-art schedulers. In fact, classical approaches both in Artificial Intelligence and Operational Research aim at optimizing a given objective function. These approaches, and the solutions they find, are based upon the hypothesis that it is possible to determine *a priori* all the aspects of the problem. For this reason, efforts toward the optimization of classical objective functions might turn out to be useless once no-deterministic problems are considered.

The main thesis of this dissertation is that flexible solutions turn out to be useful in hedging against uncertainty. Flexible schedules are solutions that are able to provide a fast answer to external and/or internal changes (i.e., changes that stem from the executional environment or problem updates). This approach is specifically tailored for problems for which the execution is uncontrollable a priori. A flexible solution will retain a set of solutions to answer to the different contingencies. Of course, to guarantee fast recovery during execution, it is necessary to go beyond this definition but also include mechanisms which will guide from one solution to another when the current solution is no longer consistent.

1.1 Project Scheduling

The project scheduling problem is defined in literature as the assignment of start and end times to a set of tasks (or actions), which are generally constrained among each other. Constraints are typically either time constraints or resource constraints. Coordination of production in a factory, management of space missions, and transportation scheduling to support crisis management are representative examples. However, feasibility alone is seldom the goal of scheduling: in fact usually the scheduling action has to be performed together with the satisfaction of a set of objectives and preferences. Therefore, the scheduling problem is primarily concerned with figuring out *when* tasks should be executed so that the final solution guarantees "good" performance through the optimization of given objective functions.

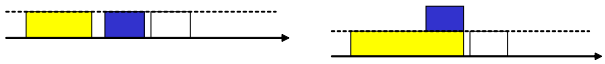
Real-life scheduling problems, like those in industrial applications, typically involve constraints that are often wide ranging and complex in nature. In manufacturing production environments for example, resource allocation decisions must be consistent with capacity limitations, machine setup requirements, etc. Similarly, production activities have associated durations and precedence constraints, and may require the availability of multiple resources (e.g., machines, operators, tooling, raw materials). For these reasons, scheduling problems are generally very complex and it can be shown that many of them are NP-hard. For the purpose of this dissertation, we can recognize the scheduling problem as composed of the following elements:

Activities: $A = \{a_1, ..., a_n\}$ is the set containing the jobs, operations or tasks. All the tasks have to be executed in order for the schedule to be completed. Every activity is characterized by a processing time p_i.

Resources: $R = \{r_1, ..., r_m\}$ is the set containing the resources required to execute the activities. Execution of each activity a_i can require an amount req_{ik} of resource r_k during its processing. There are different kinds of resources: disjunctive or cumulative, renewable or consumable, among others[1].

Constraints: The constraints are rules or restrictions that limit the possible allocations of the activities. They can be divided into two types: (1) the *resource constraints* limit the maximum capacity of each resource. For example, there may only be a certain number of machines or people available to work on some activities at any given time. (2) the *temporal constraints* impose limitations on the times at which activities can be scheduled. A unary constraint restricts a single activity, usually with a release time or a deadline. A binary constraint is imposed between to activities, for instance in order to bind the instant of occurrence of their start times.

[1] Along the dissertation we will consider only cumulative, renewable, resources: when an activity finishes, the resources it used become available for other activities. Common examples of renewable resources include machines, equipment and people.



(a) Initial allocation of three activities on the same resource

(b) A resource conflict arises when the first, yellow, activity lasts longer than expected during the execution

Figure 1.1: Brittleness of the classical fixed-time schedules

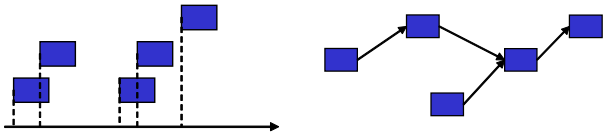
Because it has so many real-world applications, the scheduling problems has been widely studied by many scientific communities, such as the Artificial Intelligence (AI), Management Science (MS), and Operations Research (OR) community. Systematic approaches search for optimal schedules. Although they are able to achieve the highest solution quality, they rarely scale well and are limited to problems of limited size. Heuristic algorithms, instead, sacrifice optimality to find good solutions in a reasonable amount of time. Finally, local search is a heuristic approach that improves solutions by iteratively considering their "neighborhood" in the search space to obtain a better solution.

Approaches to project scheduling can be also subdivided in constructive and iterative repair. Iterative repair methods differ from the constructive methods in the fact that they begin with a first complete, possibly flawed, solution (or set of assignments) and then they iteratively modify (or repair) the solution to obtain a consistent schedule. Constructive scheduling methods, instead, incrementally extend a valid partial schedule (in terms of assigned activities) until a solution is achieved[2].

1.2 Scheduling with Uncertainty

The main motivation behind the work described in this dissertation stems from the limitation of classical scheduling approaches. In fact, 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. 1.1. The figure shows the allocation of three different activities on the same machine. The solution associates exact time

[2]Of course both the methods can require backtracking.



(a) Fixed-time schedule: each activity is anchored to a precise start time.

(b) Partial Order Schedule ($\mathcal{POS}$): each activity is free to be allocated in the interval defined by the temporal graph.

Figure 1.2: Partial order schedules vs. fixed-time solutions

instants in which the activities have to start and end. Even though this kind of schedule represents a classical solution for a scheduling problem, it is very brittle in case the definition of the problems change during the execution. In Fig. 1.1(b) we have that if the first activity lasts more than expected this will produce a conflict in the usage of the machine requiring a new solution.

As we have seen part of the schedule brittleness problem stems from reliance on a classical, fixed-time 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 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 executional uncertainty. These are the reasons behind the definition of the *Partial Order Schedules*, or $\mathcal{POS}$. This consists in a set of feasible solutions for the scheduling problem that can be represented implicitly by a temporal graph, that is, a graph in which any activity is associated to a node and temporal constraints[3] define the order in which these activities have to be executed. It is worth noting that in case of changes of temporal aspects of the problem, like activity durations or release times, this representation allows to move from the current, flawed, solution to a new, consistent, one by simply propagating the change over the temporal graph. And this is possible with polynomial algorithms (see for instance [Dechter *et al.*, 1991]). This turns out to be a relevant property because it guarantees a fast answer to unforeseen events.

To generate partial order schedules in this dissertation we describe two methods both based on the *Constraint Satisfaction Problem* (CSP) paradigm. This has been chosen because it allows to model scheduling problem with complex constraints and

[3]Temporal constraints are binary constraints, that is constraints defined on at most two variables (nodes).

because a consistent amount of efficient solving algorithms has been described in the literature (see for instance [Baptiste *et al.*, 2001]). Constraint satisfaction and propagation rules are successfully used to model, solve and reason about many classes of problems such as scheduling, temporal reasoning, resource allocation, network optimization and graphical interfaces. The first approach implemented consists in an iterative repair method that at each step of the solving process considers the set of all the possible temporal solutions and analyzes this set by computing resource usage bounds (these bounds are obtained using the resource envelope concept introduced in [Muscettola, 2002]). Therefore the current situation will be repaired until a partial order schedules is achieved.

A second, less intuitive approach to obtain flexible solutions is based on a two step procedure. In the first stage, a classical fixed-time schedule is searched for. Hence this solution is used to obtain a $\mathcal{POS}$ via method called chaining[4]. Essentially, the approach exploits the idea of splitting the solving phase from the robustifying step. In the remainder we also refer to this approach as Solve & Robustify.

Along the dissertation we will describe different implementations of the two approaches from simple greedy search to more intensive methods. To have an evaluation of the solutions generated we also have used different and complementary measures (Sect. 3.5). The results that have been obtained will show that the second approach, even though it is not direct, dominates the first one. One of the reasons for this behavior stems from the fact that the knowledge given by the resource usage bounds can overwhelm the solving process, reducing the overall performance.

1.3 Organization of the Thesis

The dissertation is split in two parts. In the first part the problem of scheduling with uncertainty is taken under consideration analyzing the current scenario and, therefore, the *partial order schedule*, $\mathcal{POS}$, is introduced. The second part instead aims at describing different methods to build such flexible solutions.

More in detail the various issues are organized as follow: Chapter 2 presents an analysis of the current research scenario concerning the construction and the management of scheduling problem solutions under uncertainty. Starting from two opposite directions, on-line and off-line approaches, different methodologies are described and faced off. The goal is to have not only a mere list of the different works but to rather provide a critical analysis of the current state of the art.

Chapter 3 concerns the description of the robustness topic and the introduction of the flexible solution paradigm for scheduling problems. The aim of the solution described is the generation of schedules that offer some robustness in the face of a dynamic and uncertain execution environment. This gives rise to the introduction

[4]The method has been obtained by generalizing an initial proposal made in [Cesta *et al.*, 1998].

of partial order schedules. Moreover, along the chapter different properties of this solution are described.

To provide a generic framework for generating $\mathcal{POS}$s in Chapter 4 a constraint-based approach to solve a scheduling problem is described. As mentioned before the framework is based on the Constraint Satisfaction Problem paradigm, CSP. Effective methodologies based on this paradigm, can be obtained for both modeling the problem knowledge and guiding search to a solution.

In Chapter 5 we define a first approach to produce Partial Order Schedules. The method is based on a well-known paradigm called *Least Commitment*. The basic idea behind the least commitment approach is to reduce as much as possible the commitment implied by a decision. This is reflected in two aspects of the search strategy: postponing all no-necessary decisions as much as the search procedure allows it, and choosing the least commitment decision once one has to be taken.

In Chapter 6 a different approach to build partial order schedules is introduced. This method consists in building a flexible solution starting from a classical – fixed-time – schedule, where a start time value for any activity is defined. We show how starting from a single solution of the scheduling problem it is possible to apply a procedure which generates a set of solutions in the form of a Partial Order Schedule ($\mathcal{POS}$). This approach has been motivated by the need to exploit the characteristics of the fixed-time solutions in a flexible solution. In fact, maintaining the optimality while a partial order schedule is generated, can yield an appealing result. We proved that the procedure used to obtain a flexible solution maintains the original fixed-time solution among the set of schedules described by the $\mathcal{POS}$. This assures that in the best case possible (i.e., no disruptions happen) the characteristics of the fixed-time solution are preserved.

Chapter 7 provides a final analysis and discussion of the partial order schedule paradigm as well as the approaches used to produce these solutions. Finally, in Chapter 8, we conclude summarizing the results obtained and discussing both the pros and the cons of the various methods. Furthermore, we will suggest possible directions for future work.

1.4 Contributions

The main contributions of this dissertation can be briefly summarized in the following points:

- An analysis of current research contributions to scheduling with uncertainty. Different state-of-the-art approaches are described. In particular we provide a classification of the various approaches into a two-dimension framework, on-line & off-line contributions, considering the various impact of the single method on the two phases.

- Definition of a Partial Order Schedule as an implicit representation for flexible solutions. This kind of solution aims at satisfying the reactiveness requirement during the execution of a schedule. In practice, through a quick reaction[5] it is possible to switch from one solution to an other.

  Even though the reaction method present in partial order schedules is oriented to give fast answers, the minimality of these repairs may also guarantee the preservation of both the stability and the qualities of the initial solution.

- A Least Commitment Approach. Using a complete computation of the resource bounds this method represents the first least commitment approach to scheduling problems. The resource usage bounds are computed integrating the resource envelope described in [Muscettola, 2002] into a constraint based framework. The integration gives rise to both a method for analyzing the bounds, and to techniques to compute them incrementally [Policella *et al.*, 2004b] and then speed up the solving process.

- A two-step procedure: Solve and "Robustify". This approach can be viewed as orthogonal to the previous one. In fact, in this case the aim of producing flexible solutions (i.e., $\mathcal{POS}$), is split into two consecutive steps. In a first step a state-of-the-art scheduler is used to compute a feasible fixed-time solution. Hence, the solution of the previous step is used as starting point to obtain a partial order schedule.

[5] A polynomial algorithm with respect the number of activities in the schedule.

Chapter 2

Scheduling with Uncertainty Current Research Scenario

In the previous chapter we have described the project scheduling problem and the issue of scheduling with uncertainty. This chapter pertains the analysis of the current approaches present in literature concerning the construction and management of scheduling problem solutions with uncertainty. The described methodologies are classified according to different aspects: the efforts for producing baseline schedules – essential in project scheduling – and the use of recovery techniques to keep up with real-time execution. The goal is to have not only a mere list of different works, rather, to provide a critical analysis of the current state of the art. The works presented do not represent an exhaustive list; rather, they describe the realm of approaches to hedge against scheduling uncertainty. Moreover, this classification is also essential to define our approach, as presented in the remainder of this thesis.

2.1 From Scheduling Theory to Real Applications

The application of scheduling theory and scheduling systems to real domains remains a fundamental issue. In a work of some years ago, [Mc Kay *et al.*, 1988], the authors, after conducting a field study on the job shop problem, have thoroughly ascertained the extent of the actual impact of the application of scheduling theory and automated system in real cases. They claim that the characteristics of real scheduling problems can hardly be faced using a theoretical scheduling approach.

Indeed, there are a number of issues that can produce a big gap between theoretical studies and actual applications. One of these, concerns the lack of expressiveness in the definition of the problem. The problem lies in the presence of several sources of information that are often difficult to integrate in order to produce a unique and

complete set of data, due to the fact that the related information may be ambiguous, outdated, incomplete. This aspect can produce scheduling solutions that do not take into account the real nature of the problem, therefore generating schedules which are in fact unusable for practical purposes.

Another aspect is the "inherent" instability of the scheduling problem – there is always something happening unexpectedly in the shop, and the schedule is seldom stable for longer than half an hour. Uncertainty in scheduling may arise from several sources: machine breakdown, unforeseen arrival of new orders, changes in existing orders, modification of release dates and deadlines, uncertainty in the duration of activities, etc. Therefore, the dynamic nature of a real environment, does not allow to take decisions based upon rigid assumptions about real world behavior. This implies that in real environments it may not generally be useful to spend significant efforts to produce optimal solutions, since the optimality is achieved only if the solution can be executed as planned. On the contrary, a sub-optimal schedule that contains some built-in flexibility for dealing with unforeseen events, might provide useful characteristics for the execution.

A different aspect to observe is that in real applications, several tools are used to achieve a solution for the scheduling problem. This provides a major flexibility to the scheduler which for instance might be able to alter both the short-term and the long term processing logic. Unfortunately these means are not formalized in the scheduling theory, due also to their inherent complexity. Hence, they are not taken into consideration during the scheduling resolution phase. This issue represents a limit in the ability of tackling the problem. Moreover due to the great number of available tools, human schedulers may be able to produce better solutions than an automatic scheduler, increasing the skepticism toward scheduling theory techniques.

An important observation we can find in [Mc Kay *et al.*, 1988], is that to bridge this gap it is necessary that both scheduling theory and real domains make some steps towards each other. On one hand, in real domain applications like factories, it is necessary to reduce the variability of the different characteristics. On the other hand, scheduling theory should extend its scheduling models to describe and understand how schedulers actually behave.

In the last decades, several efforts have been made in scheduling theory and systems, toward different directions. Different systems have been actually applied to real domains (i.e., ILOG Scheduler, CHIP), and most of the research work described so far has been developed since then. Nonetheless, as it has been pointed out in [Montana, 2002; Smith, 2003], the scheduling problem is all but a solved problem, and the research on the deployment of scheduling for real domain applications is still a challenging and a relevant issue.

In this chapter, one of the most significant aspect among those creating the gap between the scheduling theory and real world applications is taken into account: scheduling with uncertainty. This involves the analysis of different dynamic issues that can arise during the execution of a schedule, from the aspects related to the lack

in the definition of the problem, to the occurrence of unforeseen events.

2.2 A classification of current trends

In a scheduling problem the project activities are subject to precedence constraints (which can represent causal ordering or technological needs) and resource constraints. Most of research in project scheduling has concerned the study and the development of procedures to generate optimal solutions assuming a deterministic environment and complete information. However, during the actual execution, the components of a scheduling problem have seldom a complete predictable behavior. Even though an initial solution can be built on the basis of expected values and/or statistical analysis, the evolution of the environment is hardly foreseeable and it is often necessary to "fight" against adverse events.

For these reasons, in the last years, the literature regarding scheduling with uncertainty is growing rapidly. Different contributions come from both Operations Research, Management Science, and Artificial Intelligence. These techniques try to answer the different needs which arise from different sectors, from space to industrial applications. The different methodologies for scheduling with uncertainty can be categorized as follows:

Synthesis of Robust solutions: the approaches that fall in this category exploit the knowledge about possible and uncertain events in order to produce schedules able to absorb some amount of execution unpredictability.

Partially Defined Schedules: these approaches usually define a partial order of the scheduling tasks and use such a flexibility to hedge against unforeseen events. In this case no uncertainty knowledge is used.

Rescheduling: in this case the current schedule is adjusted on-line, once an unexpected change does not allow the execution [Smith, 1994a; El Sakkout and Wallace, 2000]. Among these approaches, it is possible to make a further distinction between the methods which try to maintain the repair as local as possible to preserve the stability of the solutions, and those which exploit the possibility of a more global repair to obtain better quality schedules.

Dynamic Scheduling in this case there is no baseline schedule. At any time interval during the execution, the next set of activities to be executed is selected according to specific policies and/or priorities.

In order to organize the different approaches to scheduling with uncertainty in a logical schema, we consider the two main phases of the schedule life cycle:

Off-line phase: the goal of this stage is to build an initial schedule that is the reference point for the next execution. In practice, here a predictive schedule is

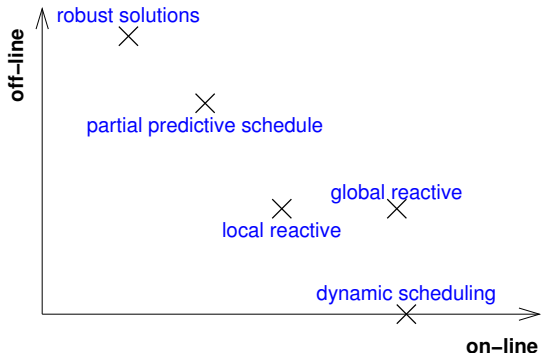


Figure 2.1: Classification of some approaches with respect to the off-line and the on-line phases that are involved.

computed based on the current knowledge of the problem and of the possible evolution of the execution.

On-line phase: The following step aims at managing the actual execution of the problem while hedging against possible, unforeseen events which stem from both the lack of sufficient information about the problem and the unpredictability of the external environment.

Figure 2.1 shows a classification of different approaches according to the off-line and on-line phases. Both axis represent a qualitative estimation of the efforts on such directions rather than an exact quantitative value. It is possible to see that, as mentioned above, in this space the dynamic scheduling approach lies on the on-line axis since it does not use any predictive solution (i.e. there is no off-line effort). Continuing the description toward increasing values of the efforts in the off-line phase, we find the reactive scheduling approaches. The two types of reactive approach, local and global, require different on-line efforts since they involve respectively a subset of activities of the whole problem. Next, we find the approaches that produce partially defined solutions. In general their flexibility guarantees minor requirements during execution than in the case of reactive approaches. The methods that require a more expensive off-line phase (trying to assure a minor on-line impact) are those which build robust solutions. In this case there is a greater effort during the off-line phase to consider the knowledge about uncertainty in the solving process. We would also like the reader to notice that no approach can lie on the off-line axis because, as mentioned before, this would require an ideal, unachievable, situation. Therefore, it is possible to note that considering the two phases it is possible to classify the different works. In fact there exist approaches that, though they present a predominant on-line (off-line) aspect, they also present an off-line (on-line) phase. In other words, we

show that any approach can be represented by a combination of a construction and a management phase, and moreover, by exploiting the ratio between the two steps it is possible to have a classification of the different works.

A further aspect can be recognized considering solely the on-line phase. The goal of this phase is to monitor the execution of the solution and in case of need, to recover the situation. This can be modeled as a *Rescheduling problem*. In fact, given a project schedule (eventually empty), the on-line phase of a solution consists in managing both the changes in the scheduled activities (resources) and the unforeseen events that modify the environment. This implies the presence of an execution manager with the task of monitoring the evolution of the schedule. The presence of such an execution manager introduces a further classification of the different approaches according to the characteristics of the possible repair actions during the on-line step. The following items represent the two opposites of the spectrum:

- approaches which require an execution manager which is able to reason about the changes and to synthesize an adjustment (new solution): it is the case of both dynamic scheduling and reactive approaches.

- Approaches in which the execution manager only has to switch toward a new "pre-defined" solution. The repairs in this case simply propagate the changes over the current situation.

It is worth noting that in the first case the solution is updated during the actual execution, whereas in the second case, the solutions consists of a set of schedules which can be represented implicitly.

To conclude, in the life cycle of a schedule (off-line, on-line phases), the execution phase is not only a consecutive step but mainly a complementary step to the construction phase and vice versa. For this reason it is possible to find a dependency between the predictive method chosen to construct a baseline schedule and the reactive approach necessary to assure its execution. In general, there are different factors which can suggest a more developed predictive phase or, on the contrary, a more sophisticated on-line management:

- a major effort in the reactive method could be necessary because of the unforeseeable nature of the events;

- a major effort during the execution phase is also useful to avoid redundant baseline schedules and to exploit the ability to find (sub-)optimal solutions;

- on the contrary, if the reaction methods are time consuming, it is preferable to make a greater effort on the construction of the baseline schedule. This is necessary in order to have a low impact during system reconfiguration.

For the above reasons, important aspects in the choice of the method are both the complexity of the scheduling problem and the complexity of the execution management of a schedule (i.e., the complexity of the rescheduling problem). For instance, a

completely reactive approach is more appropriate for problems where no constraints exist between pairs of activities than for problems where time windows may define complex relations among activities.

2.3 Analyzing Different Approaches

In this section different approaches to scheduling with uncertainty are described. Particular attention has been given to those works based on the construction of baseline schedules. In fact, in most real applications, the construction of these initial schedules is recognized as central; it allows the anticipation of potential performance obstacles (e.g., resource contention) and provides opportunities to minimize their harmful effects on the overall system behavior.

We start by introducing different techniques used to achieve robust schedules. Section 2.3.2 is dedicated to the description of different reactive approaches. In what follows, we consider different approaches that try to achieve a compromise between the advantages which either predictive or reactive approaches may give. In Sect. 2.3.5 other approaches are described. These approaches highlight interesting aspects which may give directions for future works.

Of course, we do not claim to provide a complete list of works, but our efforts are aimed at mentioning the main directions. Furthermore, we analyze more in detail those works that are more connected to the goals of our research. Different complementary surveys on this topic have been published lately, for instance the reader can refer to [Davenport and Beck, 2000; Vieira *et al.*, 2003; Herroelen and Leus, 2004a; Aytug *et al.*, 2005].

2.3.1 Synthesis of Robust Schedules

The idea of robust schedules consists in solutions that can tolerate a certain degree of uncertainty during execution. In other words, they should be able to absorb dynamic variations in the problem due to both external reasons (exogenous events), and internal reasons (false definitions in the problem). As previously stated, the fundamental assumption of these approaches requires a certain knowledge of the possible evolution of the execution.

An intuitive approach to obtain robust schedules consists in adding redundancy to the solution. A first example is represented by [Leon *et al.*, 1994]: here a genetic algorithm (GA) for producing robust schedules in the case of job shop problems is described. The authors define an evaluation function, used in the algorithm to synthesize robust solutions, according to the actual makespan of the schedule during the execution, $M(s)$, and the schedule delay $\delta(s)$ (i.e., the difference $\delta(s) = M(s) - M_0(s)$, where $M(s)$ is the executed makespan and $M_0(s)$ is the pre-schedule makespan). The

evaluation function takes the following form:

$$R(s) = rE[M(s)] + (1-r)E[\delta(s)] = E[M(s)] - (1-r)M_0(s)$$

where $E[]$ represents the expected value and r is a real value weighted in the interval $[0, 1]$. The authors demonstrate that the value of $R(s)$ can be easily computed for schedules with only one disruption. Since an exact calculation of $R(S)$ is generally intractable[1] the authors propose three surrogate measures to overcome the problem. The simplest to compute is defined as follows:

$$RM3(s) = M_0(s) - \frac{\sum_{i \in N_f} slack_i}{|N_f|}$$

where N_f is the set of activities executing on fallible machines and $slack_i = lst_i - est_i$ denotes the slack time of any activity a_i. In order to test the solutions obtained by this approach, the authors propose a simulation which demonstrates that the mean activity slack is as good a predictor of $E[\delta(s)]$ as the surrogates become more sophisticated, and it performs even better than the exact computation of the expected delay for the single disruption case, when only one machine in the shop is fallible. Furthermore, it is also shown that using a linear combination of $RM3(s)$ as an optimization criteria, it is possible to find schedules with smaller expected delays as well as a low worsening of the makespan value with respect the value obtained using the very makespan as the optimization criteria (i.e., it is able to keep low the makespan value).

Further techniques which are based on the use of genetic algorithm are described in [Jensen, 2001; Sevaux and Sörensen, 2002]. In the first, the author studies the quality of baseline schedules using a neighborhood based robustness measure. The function takes the form:

$$R_P(s) = \sum_{s' \in \mathcal{N}_1(s)} \phi(s, s') P(s')$$

where $\mathcal{N}_1(s)$ is the neighborhood of the solution s (the set of schedules achievable by swapping two consecutive operations in the solution s), $\phi(s, s')$ is weighting function (it is $\frac{1}{\mathcal{N}_1(s)}$ in this case), and $P(s)$ is a scheduling metric (i.e., lateness or tardiness). Scheduling was done using a genetic algorithm which tends to optimize the robustness criterion. The validity of the solutions achieved with this method has been tested in an empirical framework. In this context, rescheduling has been accomplished using several methods: right-shifting, $\mathcal{N}_1$-based scheduling (in which the execution switches to the best solution in the neighborhood), hill-climbing, partial and complete rescheduling. The experiments have proved that starting from a robust schedule it is easier to have a good behavior in spite of possible, unforeseen, events.

[1] Since the effect of any event depends upon the effect of the previous ones.

In [Sevaux and Sörensen, 2002] a genetic algorithm for a single machine scheduling problem is modified in order to find a robust solutions. The authors show how the genetic algorithm can be used to find solutions which are able to preserve the quality of the initial schedule and to avoid nervousness in the project[2]. To allow the genetic algorithm to find robust solutions, the following robust evaluation function is used:

$$f^*(s) = \frac{1}{m}\sum_{i=1}^{m} c_i f(s, S_i(P))$$

where $f()$ is an evaluation function for the scheduling problem, and $S_i(P)$ specifies a particular derived solution from the initial one. Moreover, c_i is a weight and m is the number of possible derived solutions to evaluate.

Different techniques that concern the use of temporal slack are described in [Davenport *et al.*, 2001]. Here the authors describe an approach to construct robust solutions by inserting temporal slack, that is, by associating to each activity a temporal slot greater than their nominal duration. The aim is to provide the ability to absorb some level of unpredicted events without rescheduling (where for rescheduling we mean any mechanism beyond the simple shift). Rather than redefining the activity durations by adding a slack interval for each of them, the authors modify the definition of the problem including a minimal amount of slack for any activity. The method, named Time Window Slack (TWS) by the authors, allows to reason about temporal protection during the solving process. They define this value for an activity a_i as it follows:

$$slack_{a_i}(R) \geq \frac{\sum_{a_j \in Acts(R)} dur_j}{\mu_{tbf}(R)} \mu_{dt}(R)$$

where $Acts(R)$ is the set of activities executing on resource R and $\mu_{tbf}(R)$ $\mu_{dt}(R)$ are respectively the mean time between two consecutive failures of the resource R and the mean down time of resource R.

Moreover, in a further improvement they consider that the probability of unforeseen events may depend on where an activity is scheduled along the temporal horizon (Focused Time Window Slack, FTWS). To implement this new method they introduce the probability that the n unexpected events occur before or at time t: $P(N(\mu(n), \sigma(n)) \leq t)$ where $\mu(n)$, $\sigma(n)$ are respectively the expected time in which the nth breakdown occurs and its deviation. Based on this distribution, the new slack value is:

$$slack_{a_i}(t, R) \geq \sum_{n=1}^{M} P(N(\mu(n), \sigma(n)) \leq t) \mu_{dt}(R)$$

where M is a large number representing the maximum number of possible failures that may occur during execution of a solution. In the provided simulation, the two

[2]The authors introduce two types of robustness: quality and solution robustness. This aspect is taken into account again in the next chapter.

methods show a better behavior than the simple temporal slack method with respect to the tardiness metric.

An operations research approach for producing robust schedules is presented in [Leus, 2003; Herroelen and Leus, 2004b]: the authors introduce a comparison of different methods to obtain robust solutions that are able to minimize the expected deviation in activity start times. They present a mathematical model for defining this problem and highlight that this implies a dual model that corresponds to a minimum cost network flow problem, which can be solved efficiently. To have a comparison of this approach the authors have also adapted two different methods: the float factor model [Tavares *et al.*, 1998] and the linear programming based heuristic, LPH, [Mehta and Uzsoy, 1998]. The former increases the earliest activity start time by the total float of the activity and a factor $\alpha \in [0, 1]$, to face the risk of a project, while the LPH method developed by [Mehta and Uzsoy, 1998] introduces idle time in case of job shop scheduling. The results presented in [Herroelen and Leus, 2004b] show that the network flow based procedure outperforms both the adapted methods.

In conclusions, it is necessary to remark that, even if the redundancy-based method can increase the robustness of the solutions in terms of tolerance to external events, the use of this technique unfortunately produces sub-optimal solutions. Even though an activity can be anticipated in case the introduced slack is not necessary during the actual execution, this does not work in case the initial solution is used to plan external activities like material procurement. In this light the solution proposed in [Herroelen and Leus, 2004b] seems to be more plausible, although it can imply a major system nervousness.

Supermodels & Supersolutions

A particular class of robust schedules which have been investigated lately is the one which stems from the *supermodels* concept introduced in [Ginsberg *et al.*, 1998]. In the following, we first introduce the original work and then we describe a further work which extends the original idea to different domains among which, the job shop scheduling domain.

The supermodel is a class of models in which any element can represent a robust solution and moreover it allows to quantify its degree of robustness. The authors highlight how the ability to allow fast and small repairs is an essential factor in the definition of a robust solution. This model is defined in this work for the SAT and the Random3SAT problems. They define an *(a,b)-supermodel* as a model such that if the values of the variables into a set of cardinality equals to a change, then an other model can be obtained changing the value of at most b variables in the disjoint set.

The previous work has been lately extended in the area of Constraint Programming and its possible applications in the case of the job shop scheduling problem [Hebrard *et al.*, 2004b; Hebrard *et al.*, 2004a]. The authors also apply the fault tolerant concept previously described in [Weigel and Bliek, 1998]: a solution S for a CSP

is said to be be *fault tolerant*, iff for a value a for any variable X of S, there exists another variable value b in the domain of X, such that S remains a solution replacing a with b. Based on the previous definitions in [Hebrard *et al.*, 2004b], the concept of *(a,b)-supersolution* for a CSP has been defined as a solution of the problem such that the loss of the values of at most a variables can be repaired by modifying the assignments for these variables and at most b further variables. The authors concentrate their attention on the $(1, b)-$supersolutions, and also introduce a first algorithm to find them as well as the most robust optimal solutions. They also consider the job shop scheduling problem, where it is possible to obtain more robust solutions without sacrificing the makespan.

The idea behind supermodels and supersolutions is interesting and can produce significant results. Nevertheless, we want to underline two possible issues which can limit its development. The first, as mentioned in the original works, is the complexity of finding supermodels (supersolutions). For instance, in [Hebrard *et al.*, 2004b] it is proved that deciding if a CSP has an $(a, b)-$super solution is NP-complete for any fixed a. The second aspect is that both supermodels and supersolutions are based on a qualitative estimation of the problem. By definition, this estimation considers only the number of changes and repairs and not their magnitude. If this approach can work for problems like SAT (boolean value, the change is always of magnitude 1) and job shop scheduling (only precedence constraint are present, hence it is always possible to find a solution), it may present some limitations on more complex problems. For instance, the number of repairs can change according to the magnitude of a change. In general we think it is necessary to add a third dimension, beyond the number of changes and repairs, to bound the magnitude of their impact. This approach might also contribute to identifying tractable sub-problems.

2.3.2 The Rescheduling Problem

In this section, two significant methods for a reactive approach toward scheduling uncertainty is described. Unlike robust schedule methodologies, such approaches do not consider any uncertainty source during the construction of the initial solution, but they rather concentrate their efforts in the on-line phase, where the schedule is adjusted once a change compromises execution. The approach is based on the implicit assumption that it is not generally possible to bound the scope of change required to the current schedule in advance.

In [Smith, 1994a; Smith, 1994b] the author describes OPIS, a scheduling system designed to incrementally revise schedules in response to changes in solution constraints[3]. This system implements a constraint-directed approach[4] to reactive

[3]OPIS, which stands for OPportunistic Intelligent Scheduler, has been developed originally for manufacturing production scheduling at Carnegie Mellon University.

[4]For a brief introduction to constraint-directed search and to the constraint satisfaction problem

scheduling. Constraint analysis is used to prioritize outstanding problems in the current schedule. More specifically, two types of conflicts can be detected: time and resource conflicts. This information, in turn, provides a basis for selecting among a set of alternative modification actions, which differ in conflict resolution and schedule improvement capabilities, computational requirements and expected disruptive effects.

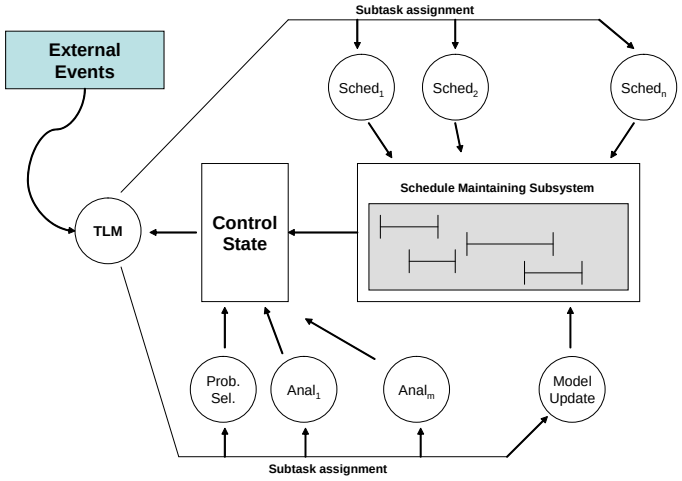


Figure 2.2: The OPIS Scheduling Architecture

Figure 2.2 schematically describes the control architecture defined in OPIS. The architecture presumes a portfolio of scheduling methods (or knowledge sources) that carry out designated scheduling tasks and make changes to a commonly accessible representation of the current solution (this can be viewed as a Constraint Data Base). An additional "model update" knowledge source is invoked upon receipt of external notification of constraint changes (e.g., new requirements, longer than expected activity durations, machine breakdowns) to reflect their consequences on the current solution. The introduction of changes to the current schedule results in the posting of control events in a global description of the system's current control state. At each instant, the control state characterizes the set of outstanding problems that remain (i.e., current conflicts in the schedule, set of commitments that remain to be made, unexplored opportunities for improving the solution). A separate set of analysis knowledge sources extends this control description to provide the information necessary to support the formulation of subsequent scheduling tasks.

The OPIS approach is based on the idea that schedule adjustments should proceed opportunistically, keeping in mind that revision actions may have unforeseen interactions which may create a domino effect over the scheduling. For this reason,

paradigm on which it is based on, the reader can refer to Chapter 4.

different recovery strategies are available: they range from generic heuristics to more specialized revision procedures, for instance, the positioning of schedule components back and forth.

It is worth noting how the idea behind the OPIS system has been applied to more complex domains, like distributed domains. For instance, its generalization, DITOPS (DIstributed Transportation Scheduling in OPIS), has been applied to problems in military crisis action deployment logistics.

A different, not local, direction has been pursued in [El Sakkout and Wallace, 2000]. Here the authors introduce a particular rescheduling problem: the *minimal perturbation problem*. It consists in a 5-tuple $(\Theta, \alpha_\Theta, C_{del}, C_{add}, \delta)$ where

- $\Theta = (V, D, C_\Theta)$ is the CSP representing the initial, deterministic, problem;
- α_Θ is one of the solutions of Θ;
- C_{del} and C_{add} are respectively the sets of added and removed constraints which model changes regarding the initial problem;
- the function $\delta(\alpha', \alpha'')$ measures the difference between two different assignments.

The objective of the problem is to find a solution for $\Theta' = (V, D, C_{\Theta'})$, where $C_{\Theta'} = (C_\Theta - C_{del}) \cup C_{add}$, such that it minimizes the value of the function $\delta(\alpha_\Theta, \alpha_{\Theta'})$. This problem has been introduced to formalize the need of minimally disruptive rescheduling. The solving process is based on the interleaving of constraint programming and linear programming. The latter has been used to find solutions for the temporal subproblem (*probes*). These solutions are then integrated into constraint programming. As a result, a complete, repair-based search is obtained (*probe backtracking*). It is worth noting that in order to minimize the disruption, or on the other hand, in order to preserve the solution's stability, for any variable $x \in V$ the equation $d_x = |x - c|$ is considered, where c is the value given to x in the assignment a_Θ. Therefore the variables d_x are added to the linear programming objective function while the linear constraints $d_x \geq x - c$ and $d_x \geq c - x$ are added to the constraint set. This allows to control the "distance" between two subsequent solutions, and the presence of the d_x component in the objective function penalizes the choice of solutions which exhibit a great difference from the starting solution.

An important aspect to consider in the reactive approach is when to apply a rescheduling procedure. Of course an intuitive solution would be to reschedule the actual solution only when a change in the problem occurs. However, in case of frequent changes this may imply a nervousness in the solution which can hardly be tolerated. Therefore it would be useful to obtain flexible solutions which can give a certain degree of tolerance in order to avoid an overly frequent rescheduling phase. Nevertheless, when to reschedule the solution remains an important issue. In [Vieira *et al.*, 2003] the authors include the timing of rescheduling as a dimension in their

rescheduling framework (rescheduling can be periodic, event-driven, or a combination of the two), while in [Aytug *et al.*, 2005] it is pointed out that the frequency of rescheduling is an important factor in scheduling performance.

Before concluding, it is worth remarking the difference between the two reactive approaches: the local approach, represented here by OPIS, is based on an analysis of the outstanding problems due to unpredicted events; in order to eliminate the different flaws, the method proceeds by triggering local modifications which can have different impacts on the whole solution. On the other hand, the approach based on the resolution of the minimal perturbation problem, pursues a global approach to solve the problem. This in general guarantees to find a higher quality solution with respect to a local method. Moreover, the method described above [El Sakkout and Wallace, 2000] allows also to explicitly preserve the original solution (stability). Even though these aspects are important, the global approach can present a lack in reactivity with respect to a local approach, requiring a greater time for system reconfiguration. Both methods present pros and cons: only after a thorough analysis of the characteristics of the particular scheduling problem, it is possible to express a preference. For instance, in order to control the execution of very complex problems, a local approach might be more indicated. This aspect is considered again in Sect. 2.4.1.

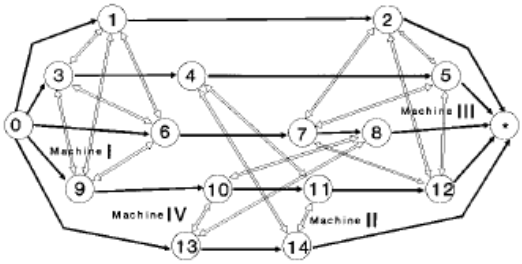
2.3.3 Partially Defined Schedules

Partial predictive scheduling consists in an intermediate approach between the use of a baseline schedule and the completely dynamic approach. Such approaches are based on the consideration that robustness can be increased by introducing flexibility in the schedule generation phase. A solution is considered flexible if it is easily repairable, hence a flexible solution allows a fast reaction. This can be achieved by computing several schedules, instead of a single and brittle one. As a result, during the execution of the schedule it is possible to switch from a solution to another in case of unforeseen events and/or changes in the scheduling problem definition.

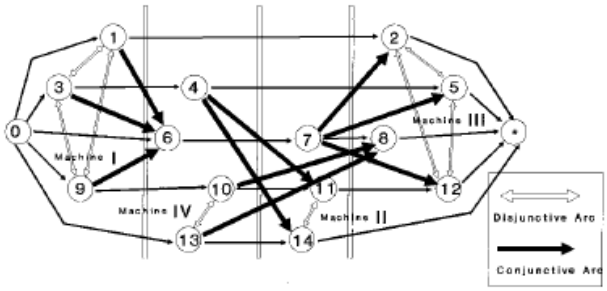
The main characteristic of this kind of approach lies in the way in which such a set of solutions is represented. In fact, an explicit representation (e.g., a set of start-time vectors) can be expensive, thus limiting the number of possible solutions. Therefore, the different approaches are singled out with respect to how the set of solutions is implicitly represented.

The idea behind the work described in [Wu *et al.*, 1999] is based on the classical disjunctive graph representation, $G(N, A, E)$, of job-shop scheduling problems and on the associated conjunctive, or directed, acyclic graph representing one of its solutions [Roy and Sussman, 1964]. The authors point out how a complete resolution of a scheduling problem can limit its flexibility. They rather propose a partial resolution of the problem which entails a decomposition of the problem into a series of ordered subproblems. The method in its turn divides the disjunctive graph into a sequence of subgraphs. This decomposition is obtained by solving some of the disjunctive arcs in

the set E, turning them into conjunctive arcs.



(a) Initial Problem



(b) Decomposed problem

Figure 2.3: Decomposition method

To better explain this concept we have borrowed Fig. 2.3 from the original work. We see that in the proposed solution (Fig. 2.3(b)) only some constraints are posted to solve some of the disjunctive arcs defined in the problem (Fig. 2.3(a)). Any single subgraph has the same configuration in the original problem, but the constraints posted create a new order among them. Based on this decomposition, the authors then introduce a *Preprocess First Schedule Later* (PFSL) scheme. This consists in a first, off-line, step in which a selected subset of disjunctive arcs are solved; in the following on-line phase, the remaining scheduling decisions are taken dynamically throughout the problem horizon. Depending on the quality of the efforts made in the preprocessing step (or on the contrary in the on-line phase) the PFSL scheme can be made arbitrarily "close" to static or dynamic scheduling.

A similar approach is proposed in [Artigues *et al.*, 2004]. Here the authors describe an ordered group assignment (OGA) representation, defined in terms of a sequence of groups for each machine, where the activities within a group are totally permutable. Therefore, during the execution phase, the decision maker is able to choose any activity in the group. The authors also compare four different dynamic

algorithms to either minimize the number of groups and maximize the number of solutions represented by the ordered group assignment.

Both the approaches presented here extend the flexibility of a scheduling solution though leaving some choices unmade. The latter is taken under consideration during the actual execution of the solution. However, this aspect may turn out to be a limitation in case these methods are extended to more complex scheduling problems. In fact, this may imply facing a hard rescheduling problem which might make it impossible to keep up the pace with the execution.

2.3.4 Managing Contingencies

In this section a different approach is introduced, concerning the synthesis of solutions which entail the possibility of particular contingencies. Like the techniques which produced the predictive partial solutions, the techniques described below provide a set of executable schedules even though they are based on the knowledge of possible disruptions modeled with a probabilistic distribution deriving from a statistical analysis.

In [Drummond *et al.*, 1994] the authors present an algorithm, named *Just-in-Case*, which produces robust schedule to face uncertainty in a real telescope scheduling domain. The most important constraint in this domain is represented by the observing windows: any observation request can be executed only in specific time intervals (or windows). The authors consider uncertainty as variability of the activity durations.

Given the importance of maximizing the usage of the time windows, a robust approach which feeds time slack in the solution turns out to produce a significant amount of wasted time. On the other hand, the use of a complete on-line scheduler can also waste time while producing a new solution. To overcome these problems the authors propose to consider the statistical models of duration uncertainty in order to produce a set of solutions or "multiple contingent" schedules starting from an initial solution. Furthermore, to avoid that the number of considered contingencies grow exponentially, the authors propose to consider only the most probable interruptions. Specifically, the algorithm is composed of the following steps: (1) the temporal uncertainty is estimated, (2) the most probable break is computed, (3) a solution that takes into account such a possibility is produced, (4) the current multiple contingent schedule is updated[5] with this new solution.

A different approach that consists in building solutions able to partially cover the set of possible contingencies is the one which stems from the application of *Artificial Immune System* (AIS) to scheduling problems. An example is [Hart *et al.*, 1998]. Here the authors propose an AIS to produce robust schedules for a dynamic job-shop scheduling problem in which jobs arrive continuously, and the environment is subject

[5] At the beginning, the multiple contingent is equivalent to the input schedule.

to changes. AISs are computational systems inspired by theoretical immunology and observed immune functions, principles and models, which are applied to complex problem domains. The authors investigated whether an AIS could be evolved using a genetic algorithm approach and then be used to produce sets of schedules which together cover a range of predictable and unpredictable contingencies. The model includes evolution through gene libraries, affinity maturation of the immune response and the clonal selection principle. Further works in this direction from the same authors can be found in [Hart and Ross, 1999a; Hart and Ross, 1999b].

The strength of these approaches derives from the use of the knowledge about possible exogenous events. However, this can also represent a limitation, primarily because it does not allow to extend these methods to not well-defined domains, secondly the methods can produce solutions that are over-fit on the statistical data and which are not able to respond in case of ignored situations[6].

2.3.5 Further Relevant Research

The approaches shown above cover part of the broader spectrum of alternatives present in literature. All these methods face scheduling uncertainty on the basis of an initial solution (baseline schedule). This solution is then updated during execution in case of unforeseen events or changes in the problem. The magnitude of the repairs depends on the way the baseline schedule is built. For instance, a flexible solution generally entails minimal disruptions after the changes have occurred.

Particular attention has been given to these approaches because of the importance retained by predictive solutions in the field of project scheduling. In Sect 2.4.2 we describe different issues that make the presence of baseline schedules essential. Anyway, there are further methods which are worth describing. These methods represent valid alternatives to consider as they might inspire new directions and improvements for future research.

Dynamic Scheduling

As we mentioned before, the Dynamic Scheduling approach consists in an on-line allocation of the activities. At each time t during the execution, a decision is taken using only the information available before or at time t. Therefore the schedule is constructed incrementally without using any predictive schedule. Decisions are taken according to scheduling policies which usually aim at optimizing a specific objective function (i.e., the expected makespan, the tardiness, etc.).

A scheduling policy describes the order in which a set of pending activities has to be executed. A first definition of scheduling policy can be given according to priority values associated with the set of activities. These values imply an order in which two

[6]The *overfitting* problem has been thoroughly studied in the area of Machine Learning.

or more competing activities have to be scheduled. A different well-known policy is the earliest-start policy which is based on the analysis of the minimal forbidden sets. A minimal forbidden set is defined as a set of activities requiring an amount of resource greater than the available resource capacity, and such that none of its proper subset is a forbidden set (the set is minimal). In this case, in order to solve the conflict, one of the activities in the minimal forbidden set has to be executed after another one has terminated its execution. The goal is then to select this "waiting" activity. Of course, the computation of earliest-start policies may be very expensive. In fact the number of minimal forbidden sets may be exponential in the number of problem activities. One way to overcome this problem is to use a priority ordering of the activities, for instance the linear preselective policies (LIN). In this case the waiting activity is the one with lower priority. There are different works which exploit these aspect as well as the construction of policies for dynamic scheduling. A compendium can be found in [Demeulemeester and Herroelen, 2002]. We also refer the reader to [Stork, 2001], where the author explores a set of algorithms to find good dynamic scheduling policies.

In [Mc Kay *et al.*, 2000] an interesting aspect of the execution of a schedule is pointed out: the physical repair following the breakdown of a machine might not correctly fix the problem. In this case, to avoid that "important" activities are adversely affected, a good strategy would be to use less critical activities to "test" the quality of the repair. To pursue this idea in [Mc Kay *et al.*, 2000] a meta-heuristic named *aversion dynamic* has been introduced. This heuristic has the effect of changing the nominal priorities to alter the sequencing of the activities, favoring the execution of less important jobs. Once the impact of having temporarily unpaired machines is not a concern, scheduling reverts to nominal procedures. In other words, the procedure pursues, for a limited time after a repair, a sub-optimal strategy. This assures that low priority activities are affected before the high priority ones. In the work the authors present the application of the aversion dynamic to the R&M heuristic [Morton and Rachamadugu, 1982]. Of course, this is not the only heuristic to apply with this strategy: any heuristic that builds the sequence in a dynamic fashion would suffice.

An approach that fits between dynamic scheduling and more static algorithm is the *Match-up Scheduling* [Bean *et al.*, 1991]. This methodology consists in adapting a preplanned schedule to a changing scheduling environment. The overall strategy is to follow the schedule until a disruption occurs. After a disruption, part of the schedule is reconstructed to match up with the schedule at some future time.

To conclude it is possible to note that the dynamic scheduling approaches permit to consider the exact evolution of the system at every instant. Unfortunately the complexity of the problem can require an expensive amount of time in order to compute optimal (or sub-optimal) solutions. Hence, in order to keep the pace with the execution, the decision process entails the use of methods which ensure a quick answer.

Sensitivity Analysis

One way to check if a given schedule is able to hedge against unexpected events, consists of applying sensitivity analysis. The approach is based on the study of possible changes of parameters according to the *what if ... ?* paradigm. If the solution tolerates a wide set of possible changes, then the schedule can be executed confidently. For instance, in [Sotskov, 1991] the author describes the stability radius, that is, the maximum amount of parameters change that maintains schedule optimality.

In [Hall and Posner, 2004] the authors introduce a systematic study of sensitivity analysis for both polynomially solvable and intractable scheduling problems. The authors start from the following typical questions: (1) what are the limits to a parameter change (or several changes) such that the solution remains optimal? (2) Given a specific change on one or more parameters, what is the new optimal cost? (3) Given a specific change on one or more parameters, what is a new optimal solution? Analyzing the preceding issues gives rise to different examples where it is possible to compute a new optimal schedule efficiently (with a low cost) by using the initial solution, as well as examples where it is worthwhile to spend additional (or different) computation to facilitate later analysis. The authors also provide some worst-case performance bounds in the case of intractable scheduling problems.

A similar approach has been described in [Ali *et al.*, 2004]. Here the authors introduce the FePIA procedure: performance (Fe)atures, the (P)erturbation parameters, the (I)mpact of perturbation parameters on performance features, and the (A)nalysis to determine the robustness. The FePIA procedure consists of the following steps: (1) describing quantitatively the requirement that makes the system robust, (2) describing the perturbation parameters, (3) identifying the impact of the perturbation parameters on the system features, (4) determining the smallest collective variation in the values of perturbation parameters that cause any of the performance features to violate the robustness requirement.

Even though these approaches are aimed at introducing different improvements to face scheduling uncertainty, they are prone to some critic. In [Wallace, 2000] the author points out that characteristics related to flexibility are not properly recognized because sensitivity analysis is based on deterministic models. These techniques are useful only as far as variation of controllable parameters is concerned.

Multi-agent Systems

Multi-Agent Systems [Weiss, 1999] represent a different approach to take into account uncertainty into scheduling problems. Indeed, this approach can be relevant, if not essential, to tackle domains where the activities which have to be coordinated are characterized by an inherently distributed nature, like large manufacturing organizations or transportations systems. In practice, each agent is responsible of the solution and the execution of a particular subproblem. Furthermore, a multi-agent system

may promote local changes to hedge against unexpected events preserving solution stability.

A first example of this kind of approach worth mentioning is described in [Ow *et al.*, 1988]. In this each resource uses activity time-bounds (a kind of flexible schedule) to assist in accommodating future activities that might become known. The CORTES project developed at CMU at the beginning of the 90's [Sycara *et al.*, 1991] consists in a decentralized, heterogeneous, multi-agent production control system. The work extends a constraint directed search within a centralized framework to a multi-agent system. The search is based on a set of "texture measures" which aim at quantifying the search space characteristics. They show how by using these measures it is possible to allow each agent to focus on its individual search space while at the same time remaining aware of the beliefs and intentions of other agents, thus bounding the impact of local decisions on global goals.

A further example is given in [Montana *et al.*, 2000] where the authors use a multi-agent approach to automatically solve a military transportation scheduling problem. This involves a set of different issues due to the large number of tasks and resources involved as well as the large number of scheduling subproblems eventually generated. The approach used by the authors consists in the implementation of a genetic algorithm for each agent. The algorithm continually creates new schedules. This process allows to periodically reintroduce diversity into the population.

2.3.6 Execution of temporal plans

Even though temporal plans are specific problems in which no resource are considered, research works on execution of temporal plans have to be mentioned. In [Vidal and Ghallab, 1996] a formalism called Simple Temporal Network with Uncertainty (STNU) is introduced in order to explicitly model plans that contain both *executable* (or requirement) and *contingent* (or uncontrollable) time-points. For such plans it is insufficient to merely ensure that the plan is consistent. It is necessary to do some off-line reasoning in order to determine if it is possible, during the execution, to make consistent scheduling decisions in any situation.

In [Vidal and Ghallab, 1996; Vidal and Fargier, 1997] three levels of controllability have been identified. *Strong controllability* if there is a fixed execution strategy that works in all the situations. *Dynamic controllability* if there is an on-line execution strategy that depends only on observed time-points and that can always be extended to a complete schedule whatever may happen in the future. *Weak controllability* if there exists at least one execution strategy for every situation.

Regarding the plans dynamically controllable, an important aspect is represented by the process of reformulating STNs into minimum *dispatchable* networks, that is, into STNs that can always be correctly executed by a dispatcher. [Muscettola *et al.*, 1998; Tsamardinos *et al.*, 1998] introduced a means to produce an efficient dispatchable network for plans that do not contain uncertainty. [Morris *et al.*, 2001]

extended this work for plans that contain uncertainty, STNUs. Specifically it presents a polynomial algorithm to perform this reformulation.

2.4 Further Remarks on Current Scenario

This section is dedicated to analyzing further issues that may play a relevant role in facing scheduling uncertainty, as well as the procedures which can be adopted for this purpose. On one hand, there is the need to consider the complexity of the scheduling problem before choosing a method. On the other hand, we highlight how the production of baseline schedules can be useful in different real domains.

2.4.1 The Complexity of Scheduling Problems

Different scheduling environments invariably present different challenges. We witness diversity related tor several aspects: in the structure of different domains (e.g., type of manufacturing system and discipline), in the types of dominating constraints, in the performance objectives and preferences which need compliance, and in the types of uncertainty which must be considered. The problem characteristics along each of these dimensions determines the modeling assumptions, the scheduling heuristics, and the most appropriate solution procedures for a successful application.

In fact the definition of these problems may give a strong bias on the validity of the approach. Two issues can arise:

- the approach may show good performance due to the relative simplicity of the scheduling considered problem;
- the approach may be tailored on an overly specific problem, and its generalization to other problems may not be so effective.

Hence, it is important to recognize these hidden aspects in scheduling problems. In the works described so far it is possible to find different examples of such “problem biases”. For instance, let us consider the supersolutions approach described in Sect. 2.3.1. In this case the authors have applied their approach to the job shop scheduling problem and defined (1,b)-supersolutions for this specific case. However this approach still considers the qualitative aspects of the scheduling problem rather than the quantitative ones. For example, in more complex scheduling problems it is not sufficient to only define the number of disruptive events but also their magnitude.

Another example is the one which arises from the analysis of reactive approaches. Here the choice between two different methodologies, local or global, depends on the scheduling problem. A very complex problem can require a large amount of time to be rescheduled by a global procedure like [El Sakkout and Wallace, 2000]. Therefore, despite their inherent sub-optimality, the approaches based on local modifications are preferable in order to keep up with execution.

2.4.2 The Necessity of Baseline Schedules

Along Sect. 2.3 we have mostly concentrated our attention toward methods which face scheduling uncertainty on the basis of an initial solution. The necessity of baseline schedules is highlighted in this section.

Previous works [Mehta and Uzsoy, 1998; Aytug *et al.*, 2005] have remarked how baseline schedules, built in a first off-line phase, have an important function for several reasons. The first reason concerns the ability of checking whether it is possible to complete the work as planned. In fact, even though the provided solution is not followed during execution, it ensures the admissibility of the planned project that turns out to be relevant in industrial practice.

A further point to note is that through an initial schedule it is possible to plan external activities such as material procurement and delivery of orders to customers [Wu *et al.*, 1999]. Furthermore it can be used to re-organize production resources in the light of an analysis of the resource requirements that can identify peaks and low capacity requirement periods. The last aspect can turn out to be very useful to increase the robustness of a solution. The visibility of future actions is of crucial importance within the inbound and outbound supply chain. Especially in multi-project environments, a schedule often needs to be sought before the start of the project that is in accord with all parties involved (clients and suppliers, as well as workers and other resources). It may be necessary to agree on a time window for work to be done by subcontractors and to organize production resources to best support smooth schedule execution. For instance large companies can share their production schedules with their suppliers in order to allow just-in-time material delivery.

A baseline schedule is also vital for cash flow projections and provides a yardstick by which to measure the performance of both management and shop floor personnel. Indeed, reliable baseline schedules enable organizations to estimate the completion times of their projects and take corrective action when needed. They allow for scheduling and resource allocation decisions that in turn should allow quoting competitive and reliable due dates.

2.5 Conclusions

Along this chapter different techniques based on scheduling with uncertainty have been described. We have seen that all the works can be classified according to the off-line and the on-line phase (see Fig. 2.1). The analysis has shown how in the current state-of-the-art there is not a single approach which emerges from the set. Rather, different approaches are more tailored for different purposes. Therefore, there is a great dependency between the single approach proposed and the kind of scheduling problem that it takes into account.

In conclusion, scheduling is a problem that occurs in a large variety of forms with

huge cumulative economic and social consequences. Proper scheduling can provide better utilization of scarce and expensive resources as well as higher satisfaction for individuals such as customers and employees. There are a few reasons why scheduling is such a difficult problem:

- the size and complexity of the search space;
- scheduling is an inherently dynamic process, schedules only remain valid for a limited amount of time. After a certain duration, the world generally has changed enough that the scheduling algorithm has to find a different schedule;
- different domains and applications require solutions of different variations of the scheduling problem.

It is our opinion that dealing with scheduling uncertainty may represent a significant step toward bridging the existing gap between scheduling theory and real domains. In fact this still represents one of the more difficult aspect in real scheduling problems.

In the next chapter we introduce our approach to scheduling uncertainty. This is based on a partial order of the activities to be scheduled. The idea is to provide enough flexibility to absorb execution unpredictability. Moreover the idea is also to provide a means to promptly answer to unforeseen events. This has been accomplished by forcing an answer with minimal disruption to also preserve the stability of the solution.

Chapter 3

Robustness through Flexible Schedules

Research efforts aimed at generating solution and quality robust schedules in combination with an effective reactive scheduling mechanism are still in a burn-in phase. In the previous chapter different approaches to scheduling with uncertainty have been shown. Among these, those based on a major effort during the construction of a partial predictive schedule (Sect. 2.3.3) may have characteristics that, in our opinion, make them preferable. For example, they seem very appropriate in the case of scarce knowledge about possible exogenous events and the need of an initial plan to better manage the scheduled activities.

This chapter describes the concept of robustness and introduces a flexible solution model for scheduling problems. Its main concern is the generation of schedules that offer some robustness in the face of a dynamic and uncertain execution environment.

3.1 Introduction

The approach described in this thesis aims at increasing the robustness by introducing flexibility in the scheduling generation phase. Flexible solutions consist in a set of possible schedules that can be followed during the execution, and, at the same time, in guaranteeing, in case of necessity, an easy and fast recovery of the current situation. In fact, as mentioned before, 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., in a factory context, due to a breakdown of a sub-set of the

available machines), or new tasks may need to be taken into account pursuant to new requests and/or goals. The approaches described in the previous chapter are based on different meanings of robust solutions. The concept of robustness, on which this work is based on, can be viewed as execution-oriented; 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 the pace with the execution guaranteeing a prompt answer to the various events.

To obtain these qualities we based 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. More precisely, our approach adopts a graph formulation of the scheduling problem and focuses on generation of *Partial Order Schedules* ($\mathcal{POS}$s). A partial order schedule is a set of feasible solutions to a scheduling problem that can be represented by a temporal graph in which at any activity is associated a node and the edges represent the constraints[1] between the activities. Within a $\mathcal{POS}$, 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 $\mathcal{POS}$ 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 $\mathcal{POS}$ – the number of fixed-time schedules (or possible execution futures) that it "contains" – is suggestive of its overall robustness[2]. In general, the greater the size of a $\mathcal{POS}$ the more robust it is. Furthermore, together with the size of the set of schedules, it is also worth considering the way in which all such solutions are distributed, i.e., how many alternatives each activity has. Therefore, the challenge is to generate $\mathcal{POS}$s of maximum possible size equally distributed over the problem.

The remaining of the chapter is organized as it follows: first we remark the different aspects that the concept of robustness entails, recalling some of the proposals that can be found in the literature. Afterward, to better specify the concept of robustness a specific scheduling problem, RCPSP/max, will be introduced. Then in Sect. 3.4 we introduce the definition of a solution paradigm: the partial order schedule, $\mathcal{POS}$. In Sect. 3.5 we also introduce two metrics which will be used to evaluate the quality of the $\mathcal{POS}$s found, and compare the different approaches described in the remainder of the thesis.

[1] In a temporal graph any constraint is, at most, binary, that is, relates at most the values of two variables.

[2] The use of an horizon, as it will be described in Sect. 3.5, is justified by the need to compare partial order schedules of finite size.

3.2 Robustness

Even though different approaches have been pursued so far, the concept of robustness for scheduling solutions, as well as in other areas, remains vague and not well defined. In fact, in the analysis of the related literature to the topic of scheduling with uncertainty provided in Chapter 2 it is possible to note different scheduling definitions of robustness with respect the different aspects that want to be taken into account. Some of these definitions have emphasized the ability to preserve some level of solution quality, such as preservation of the completion time or makespan in [Leon *et al.*, 1994; Leus and Herroelen, 2004]. For instance, in [Leon *et al.*, 1994] the robustness of a solution is defined with respect its ability to preserve the solution quality. It takes into account the difference between the expected makespan, $M_0(S)$, and the actual result of the execution, $M(S)$:

$$\delta(S) = M(S) - M_0(S)$$

A different example is given in [Sevaux and Sörensen, 2002] where the authors provide a distinction of robustness according to the following two characteristics: quality robustness and solution robustness. The first is a property of a solution whose quality does not change much from the initial one when small changes in the problem occur. The latter, instead, occurs when in the same situation (small changes) a solution does not deviate much. Then the authors conclude that the two robustness concepts can be viewed in two different spaces: the objective function space and solution space respectively for quality and solution robustness.

Alternatively, other works have considered robustness to be an execution-oriented quality. For example, in [Ginsberg *et al.*, 1998] robustness is defined as a property that depends 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. For this reason, it is claimed that a robust solution will be obtained once repairs, both small and fast, will be guaranteed. In [Hebrard *et al.*, 2004b] this idea is brought further: a solution will be robust if it is able to bound the number of changes entailed by the repair to problem changes. Finally, in [Jen, 2003] the author concentrates her attention on both the stability and the robustness aspects. She poses a series of questions: What is stability? What do stability and robustness have in common? What is robustness beyond stability? She argues that the theory of stability is not always able to capture all the characteristics that a more general robustness concept does[3].

[3] There are also web sites where these themes are being discussed. We would like to mention the informal French association of researchers in scheduling, GOThA

Based on the aspects underlined above it is possible to define a taxonomy of the different definitions according to the following aspects:

Reactiveness. An important aspect in the execution of a schedule is to promptly respond to unforeseen evolution of the external environment or to changes in the scheduling problem definition. A slow answer can compromise the ability to keep the pace with execution, and, in practice, lead to a failure of the execution itself.

Stability. The solution has to avoid an amplification of the effects of a change over all its components, creating a so called domino effect. Keeping a solution as stable as possible assures notable advantages: for instance, let us consider the case of a scheduling that involves a group of people each with different skills and different tasks to accomplish. In this case, the greater the number of people involved in the schedule reconfiguration, the greater the confusion (solution nervousness).

Solution quality. The last aspect regards the quality of a solution, i.e., makespan, weighted lateness, and so on. This point highlights the ability of the solution to preserve the performance of the initial (baseline) solution in a changing environment. Bounding the possible changes with respect to one or more quality functions is fundamental. In fact the plan has to respect "external" constraints like predefined budget for production cost or delivery due to customers.

It is worth noting the difference between the last two points. Indeed it is possible to have a great instability but the preservation of the solution quality and vice versa. Therefore, even though the two aspects seem (and often are) connected, in general they allow to consider two different perspectives of the robustness of a solution.

A final remark is that the concept of robustness is not a context free concept but is rather tightly related to the particular scheduling problem that is taken into account. In fact, even though the three aspects described above, reactiveness, stability and solution quality, are the basic ingredients of a robustness criterion, one of the three may dominate the other two according to the scheduling problem. In the case of a complex problem reactiveness will be a predominant aspect. In fact, when rescheduling is very expensive, a particular solution that allows to promptly compute a new, corrected, one can be the main need. On the contrary the reactiveness will not be an issue in the case of a manufacturing system in which temporal constraints are not so tight.

(http://www.loria.fr/ aloulou/pages/biblio_gotha.html), and the Santa Fe Institute robustness web site (http://discuss.santafe.edu/robustness).

3.3 The Reference Scheduling Problem: RCPSP/max

In this section we describe the scheduling problem on which our approach is based on. We adopt the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max, as a reference problem [Bartusch *et al.*, 1988]. 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*, p_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:

$$e_i = s_i + p_i \tag{3.1}$$

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,

$$l_{ij}^{min} \leq s_j - s_i \leq l_{ij}^{max} \tag{3.2}$$

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 3.2 and durations 3.1 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 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.

$$\sum_{s_i \leq t < e_i} req_{ik} \leq c_k. \tag{3.3}$$

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

Example 3.1 *Figure 3.1 describes a 5-activity RCPSP/max instance. Any activity a_i is composed of two time points, s_i and e_i, which correspond to its start and end time. Temporal constraints are the durations (defined between the pair of time points (s_i, e_i)) and time lags (for instance activity a_5 should not start before 3 time units*

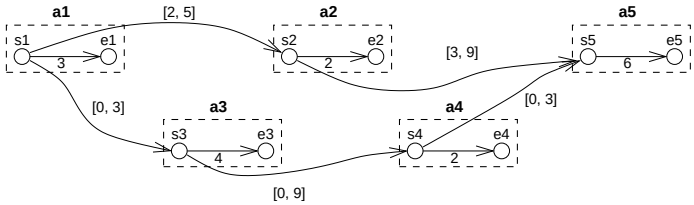


Figure 3.1: 5-activities RCPSP/max instance.

after that activity a_2 begins). Moreover we suppose that any activity requires during its processing one single unit of an unary resource r_1, that is, according to the terminology introduced above, $req_{11} = req_{12} = req_{13} = req_{14} = req_{15} = 1$ and $c_1 = 1$. Note that in general the resource capacity can be greater than one and the activities can require one or more units of one or more resources.

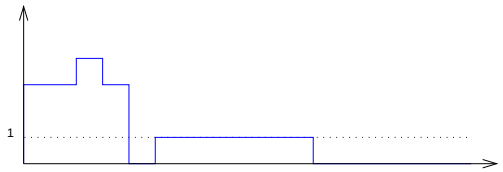


Figure 3.2: Resource profile for the temporal solution $s_1 = s_3 = s_4 = 0$, $s_2 = 2$ and $s_5 = 5$.

A first temporal solution is the following: $s_1 = s_3 = s_4 = 0$, $s_2 = 2$ and $s_5 = 5$. However if we consider the resource constraint we find a conflict in the use of the resource (see Figure 3.2). In fact this solution implies a parallel execution of more than one activity (four, a_1, a_2, a_3 and a_4, in the interval $[2, 3]$) and because of the resource capacity and the activities requirement, this is not allowable.

A different temporal solution that respects also the resource constraint is $s_1 = 0$, $s_2 = 3$, $s_3 = 5$, $s_4 = 9$, and $s_5 = 11$. This solution in fact, avoids the execution of two or more activities simultaneously.

In spite of the simplicity of the example shown above the RCPSP/max problem is a very complex scheduling problem. In fact not only the optimization version but also the feasibility problem is NP-hard [Bartusch *et al.*, 1988]. The reason for this NP-hardness result lies in the presence of maximum time-lags. In fact these imply the

presence of deadline constraints, transforming feasibility problems for precedence-constrained scheduling to scheduling problems with time windows[4].

3.4 Flexible Solutions

As introduced above, our approach concerns the production of flexible solutions. The idea is that flexibility can assure a certain degree of robustness because it assures a prompt reaction that will allow to keep the pace with the execution.

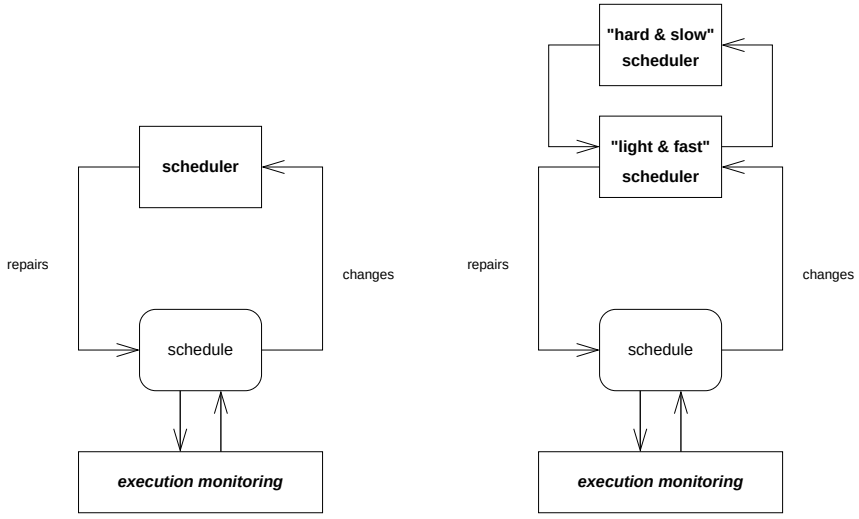
Figure 3.3(a) describes the execution of a schedule. This is given to an executor (it can be either a machine or a human being) that manages the different activities. If something happens (i.e., an unforeseen event occurs) the executor will give feedback to a scheduler module asking for a new solution. Then, once a new solution is computed, it is given back to the executor. In Fig. 3.3(a), instead, the execution of a flexible schedule is highlighted. The substantial difference in this case is that the use of flexible solutions allows to consider two separate phases of rescheduling: the first consists in facing the change by immediate means like its propagation over the set of activities. In practice, in this phase the flexibility characteristics of the solution are exploited (for this reason we named this module *light & fast scheduler*). Of course it is not always possible to face an unforeseen event by using only "light" adjustments. In this case, it will be necessary to ask for a more complete scheduling phase. This will involve a greater number of operations than in the light phase. This module has been named *hard & slow scheduling*. It is worth noting that the use of flexible schedules allows to bypass the last, more complicated, phase in favor of a prompt answer[5].

A flexible solution can turn out to be very efficient with respect to execution. Before we introduce a possible solution let us now analyze deeper the concept of flexibility. There are several aspects that a flexible solution should guarantee:

The ability of representing a set of solutions. Intuitively, having a huge set of schedules which cover all the possible evolutions of the world is the best one can

[4]The reader can notice that the makespan minimization problem of the relaxed version without maximum time lags is still NP-hard. But even more, it is among the most intractable combinatorial optimization problems. This has been proved in [Schäffter, 1997] transforming the vertex coloring problem in graphs to a special case of a resource constrained project scheduling problem with makespan objective. For an instance of vertex coloring, introduce a job j with unit processing time $p_j = 1$ for each vertex. Then, add a resource k for each edge, let $r_k = 1$ and set $req_{jk} = 1$ for the two jobs which are incident to the edge, and $req_{jk} = 0$, otherwise. The makespan of the so-constructed scheduling instance equals the minimal number of colors required to color the vertices of the graph. Thus, as for vertex coloring, there is no polynomial-time approximation algorithm with a performance guarantee less than n^{ϵ} for some $\epsilon > 0$, unless $P = NP$ [Feige and Kilian, 1998].

[5]However the reader should note that these solutions are in general sub-optimal.



(a) General rescheduling phase

(b) Rescheduling phase using a flexible solution

Figure 3.3: Rescheduling actions during the execution.

expect. On the other hand, it is also necessary to have an implicit, compact, representation of this set. An explicit representation (e.g. a set of start-time vectors) can be expensive and then limit the number of solutions (this issue has been also pointed out in Sect. 2.3.3).

A means to select a new solutions. When a huge set of alternatives is given, one question arises: which one do you have to select? To choice among different alternatives can become complex, and expensive. For this reason it will be necessary to have a mechanism, compiled in the solution, which helps the decision maker to find a new solution. Therefore the need is to have a sort of "guess" that always points the right way to follow. This aspect plays also a key role in the next point.

The ability of promptly answering to an external event. This point is tightly related with the reactiveness aspect of a robust solution. Above we have seen that good flexible solutions should be able to represent different schedules, but also they should be able to detect a new, best, solution when it is necessary. However it is also fundamental to consider how expensive it is to obtain a new solution. In some cases to keep the pace with the execution, it will be better to have a sub-optimal solution and wait less.

Preservation of robustness. The last aspect concerns the fact that several problem

changes and/or unforeseen events may happen during the execution of a solution. Therefore it is necessary that once a solution has been adjusted it continues to maintain robustness characteristics. Therefore it is necessary that the repair be as minimal as possible.

It is worth noting that the second and the third point make a distinction between the ability of finding a new solution and the swiftness of achieving it. Considering the first aspect it is important to underline that in case the set of solutions is big and the repair method assures minimal repairs, then also the solution quality, for instance the makespan, will be preserved.

Of course the necessity of an implicit representation of the solutions and the use of mechanisms that assure a prompt answer to scheduling uncertainty will require a "sacrifice" with regard to the number of solutions represented. In the next section a paradigm for flexible solutions is introduced. This has been implemented to answer to the requirements listed above.

3.4.1 Partial Order Schedule

This section provides a solution paradigm to answer to the need of finding an implicit representation of the set of schedules. To represent the scheduling problem we refer to the activity on the node representation: given a problem P, this is 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. In particular for each time lag constraint (3.2) there is an edge in the graph $(a_i, a_j) \in E_P$ labeled with the values $[l_{ij}^{min}, l_{ij}^{max}]$.

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 the following holds:

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

that is, a resource conflict is present. This subset is called a forbidden set [Bartusch *et al.*, 1988] (or contention peak), and a *minimal forbidden set* (or resource conflict) is a forbidden set $F_{min} \subseteq F$ such that each of its proper subsets is not a forbidden set[6]. A property of these is that 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

[6]The minimal forbidden sets are taken under consideration in [Laborie and Ghallab, 1995] where it is named *minimal critical set*, or MCS. We analyzed the MCS later in Sect. 4.3.1

recalling that a time feasible schedule is a schedule that satisfies all the constraints defined in 3.1 and 3.2, and a feasible schedule is a schedule that is both time and resource feasible, we can define a *Partial Order Schedule* as follows:

Definition 3.1 (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 $G_{POS}(V_P, E_P \cup E_R)$ such that any* time feasible *schedule defined by the graph is also a* feasible *schedule.*

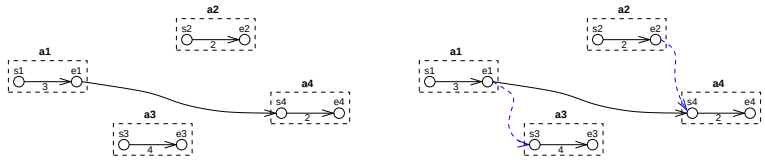
In practice a $\mathcal{POS}$ is a set of partially ordered activities such that any possible complete order that is consistent with the initial partial order, is a resource and time feasible schedule.

It is worth noting that a partial order schedule provides the opportunity to reactively respond to external changes by simply propagating the effects of these changes over the "graph", by using a polynomial time calculation. 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. To propagate all the effects of the new edge over the entire graph it is necessary to achieve the *arc-consistency* of the graph. That is, ensure that any activity has a legal allocation with respect the temporal constraints of the problem. To obtain this status of the graph it is possible to apply the Bellman Ford algorithm that finds the shortest paths from a single source vertex to all other vertices in a weighted, directed graph[7] (Single-Source Shortest Path) with a time complexity equals to $O(nm)$ where n and m are, respectively, the number of activities and the number of constraints in the problem. However it is possible to obtain better results considering incremental algorithms that avoid to compute the solution from scratch at each step. For instance [Cesta and Oddi, 2001] introduces 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. The last result can not be extended in the case where a constraint is removed from the graph. This can be the case, for instance, of the reduction of an activity duration. In this case during the execution it is possible to have two different approaches: either recompute a new solution from scratch by using a complete algorithm or simply do nothing.

It is worth noting 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. Therefore a partial order schedule provides a mean to find a new solution and ensures to compute it in a fast way.

Example 3.2 *Figure 3.4(a) represents a 4-activity problem. Any activity requires one unit of a resource r_1 with capacity $c_k = 2$. One possible partial schedule for*

[7] Note that weights may be negative.



(a) Scheduling Problem (b) A Partial Order Schedule

Figure 3.4: Example of a partial order schedule.

this problem is given in Fig. 3.4(b). Here it is possible to see that two precedence constraints are posted to avoid a over-request of the resource. Moreover any possible temporal constraint can be posted maintaining the consistency of every solution in the resulting $\mathcal{POS}$.

The example above also highlights the following interesting property:

Property 3.1 *Given a partial order schedule $\mathcal{POS}_i$, the result of posting on it one or more temporal constraints, is still a partial order schedule, $\mathcal{POS}_i^{'}$.*

This property turns out to be relevant as it assures that during its execution, the solution (i.e. the $\mathcal{POS}$) preserves robustness characteristics.

In the following, two concepts connected to the partial order schedule are introduced: respectively the earliest start schedule and the makespan of a $\mathcal{POS}$.

Definition 3.2 *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 $1 \leq i \leq n$.*

Definition 3.3 *The makespan of a partial order schedule is defined as the makespan of its earliest start schedule, that is,*

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

Before concluding we want to introduce a further remark on partial order schedules. In [Roy and Sussman, 1964] the authors introduce the disjunctive graph representation of the classical job shop scheduling problem and describe how a solution can be achieved by solving all the disjunctive constraints transforming any of these in a conjunctive one. Also in our case, solving all the disjunctive constraints is required in order to achieve a $\mathcal{POS}$. Now, the disjunctive graph representation can be

extended in a more general case where multi-capacity resources are defined. In this case "disjunctive" hyper-constraints among activities that use the same resource are introduced. Based on this representation we can note that a partial order schedule is obtained once any disjunctive hyper-constraint is solved. In this case, a set of precedence constraints is posted to solve each hyper-constraint. The number of constraints necessary is of the order of $O(\frac{n}{c_k})$ where n is the number of activities and c_k the capacity of the involved resource.

Facing different changes

Even though the characteristics of partial order schedules are tailored on temporal changes that can happen during execution, this kind of solution can turn out useful also to face other kinds of change. For instance, during execution it is possible to have the request of adding new activities and/or the resource availability can be reduced (also completely) for certain time intervals. Both these events can be modeled as adding a new activity to the problem. In fact the reduction of $red_k < c_k$ units of the capacity of the resource r_k during the time interval $[t', t'']$ can be modeled by an activity a_r with resource requirement $req_k = red_k$, start time $st_r = t'$ and duration $p_r = t'' - t'$.

Even though the start time and the duration of the activity are already defined, posting this new element does not imply a single solution. In fact, let us consider the following situation: on a resource r_k with capacity $c_k = 3$ there are three activities (a_1, a_2 and a_3) all allocated during the time interval $[3, 17]$. If we have a reduction of the capacity of one unit during the time interval $[3, 9]$ this can be represented with a dummy activity a_4 with $req_4 = 1$ that starts at $st_4 = 3$ and lasts 6. The problem is which activity among a_1, a_2 and a_3 to postpone?

The situation above highlights that unlike the case of temporal changes, the $\mathcal{POS}$ is not able to give a unique solution but it is necessary to take a decision. However, the partial order schedule providing a certain degree of flexibility can allow a fast propagation of the decision taken. One possible solution we can suggest for the situation above, is based on the consideration of the critical path. The idea is to avoid decisions that can affect the set of activities belonging to the critical path, in order to obtain a smaller disruption of the makespan value. In this light, the work of [Artigues and Roubellat, 2000] provides a starting point to consider this sort of extension.

Final remarks

Along this section different and relevant advantages introduced by the use of flexible solutions have been described. The proposed solution, which we named *Partial Order Schedule*, consists in a set of feasible solutions of the scheduling problem that can be represented by a temporal graph. This paradigm guarantees a set of characteristics that can be summarized as follows:

- the temporal graph allows to implicitly define a new solution to recover the situation after a disruptive action. In fact the temporal graph describes the partial order that the set of activities must respect. Therefore to obtain a new consistent solution it will be sufficient simply to propagate the change over the temporal graph.

- the underlying temporal graph allows also to compute the new solution quickly. In fact as said before a new solution is obtained by propagating the change over the graph and this can be accomplished by using polynomial algorithms.

- the propagation step by definition computes the minimal repairs necessary to take into account the change in input, and to obtain a new consistent solution. This minimality aspect might avoid unnecessary domino effects preserving the stability of the solution.

- the minimal repairs produced during the propagation step also allow to avoid a great "consumption" of the set of solutions represented in the $\mathcal{POS}$. This allows to preserve the flexibility (or robustness) characteristics of the solution that can be used to face further changes.

While the first two points are structural properties of any $\mathcal{POS}$, in the case of the last two, the efficiency will depend on the quality of the single $\mathcal{POS}$. In the next section we introduce two different metrics to evaluate the quality of $\mathcal{POS}$s in terms of their flexibility. A solution with great flexibility can allow to avoid domino effects: this aspect turns out to be relevant in terms of the preservation of both the stability and the flexibility of the solution.

3.5 Metrics to Compare Partial Order Schedules

This section is dedicated to the introduction and analysis of metrics to compare different partial order schedules. This represents an important aspect to analyze the effectiveness of different problem solvers.

As described before, a single $\mathcal{POS}$ represents a set of temporal solutions that are also resource feasible. Thanks to this set of schedules it is possible to provide 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 $\mathcal{POS}$. Therefore, it is intuitive that the quality of a partial order schedule is tightly related to the set of solutions that it represents. In fact the greater the number of solutions, the greater the expected ability of facing scheduling uncertainty. Furthermore, another aspect to consider in the analysis of the solutions clustered into a partial order schedule is the

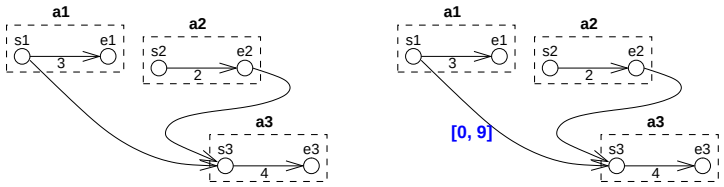
distribution of these alternatives over all the activities. This distribution will be the result of the configuration given by the constraints present in the solution.

For this reason it is necessary to use metrics that consider these aspects. A first measure is taken from [Aloulou and Portmann, 2003]. In this work the authors describe a metric, named $flex$, that can be defined as it follows:

$$flex = \sum_{i=1}^{n} \sum_{\substack{j>i \wedge \\ a_i \not\prec a_j \wedge a_j \not\prec a_i}}^{n} \frac{1}{n(n-1)} \tag{3.4}$$

where n is the number of activities and $a_i \not\prec a_j \wedge a_j \not\prec a_i$ defines a no precedence relation between a_i and a_j and vice versa[8]. This measure counts the *number of pairs of activities in the solution which are not reciprocally related by simple precedence constraints*. This allows to provide 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. The following example explains the rational behind the $flex$ metric.

Example 3.3 *Let us consider the example in Fig. 3.5 in which the labels on the edges define the time constraints between pairs of activities. Not labeled edges represent simple precedence constraints* $[0, +\infty)$*. In the figure two different problems are represented. Note that in the first there are only precedence constraints, Fig. 3.5(a).*



(a) Solution with value $flex = 0.6\overline{6}$

(b) Solution with value $flex = 0.6\overline{6}$

Figure 3.5: The two network have the same value according to the $flex$ metric even though the right-hand one is clearly worse

If the metric $flex$ is computed for both problems, we obtain the same results, that is, $flex = \frac{4}{3\times 2} = 0.6\overline{6}$. But after a deeper analysis of the two solutions it is

[8]For instance, if $a_i \not\prec a_j$ then no positive path may exist from a_i to a_j, in the graph associated with the temporal constraints of the scheduling problem.

possible to note a basic, fundamental, difference. In fact in the case in Fig. 3.5(b), the presence of the maximum constraints bounds the domain of both a_2 and a_3. These are, considering $st_1 = 0$, respectively $[3, 5]$ and $[5, 7]$.

Conversely, for the problem in Fig. 3.5(a) also fixing the value of activity a_1, i.e. $st_1 = 0$, the domains of the other two activity are still infinite, i.e., $[3, +\infty)$ and $[5, +\infty)$.

The previous example underlines an important limitation of the $flex$ metric; indeed, this is able to give only a qualitative evaluation of the solution. Even though this may be sufficient for a scheduling problem with no time lag constraints like the one used in [Aloulou and Portmann, 2003], in a problem like the RCPSP/max it is necessary to integrate the flexibility measure described above with an other one that is able to take into account also the quantitative aspect of the problem (or solution).

Furthermore, the example above also suggests to use of a temporal bound, or horizon, for the problem in order to have $\mathcal{POS}$s that represent a finite, measurable, set of solutions. In fact it is worth noting that in order to compare two or more $\mathcal{POS}$s it is necessary to have a finite number of solutions. This is possible assuming that all the 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.*. Let us consider again the example in Fig. 3.5(b): if we post a constraint $H = 10$, it is possible compute all the possible solutions, that is, the start time vectors $(0, 3, 5)$, $(0, 3, 6)$, $(0, 4, 6)$, and $(1, 4, 6)$. The goal is then to compute a fair bound (or horizon) that does not introduce any bias in the evaluation of a solution. Therefore a time interval that allows all the activities to be executed. The following points are useful to define an upper bound for the minimum completion time or makespan:

1. there is no idle time on the resources unless it is requested by the problem;
2. in the worst case, all the activities are executed sequentially;
3. the minimum distance between any pair of activities should be respected.

From these points it is possible to formulate the following horizon value:

$$H = \sum_{i=1}^{n} p_i + \sum_{\forall(i,j)} l_{ij}^{min} \tag{3.5}$$

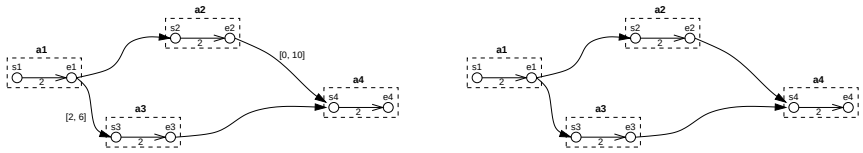
that is, the sum of all activity processing times p_i and the sum of all the minimal time lags l_{ij}^{min}. The minimal time lags are taken into account to guarantee the minimal distance between pairs of activities. In fact considering the activities in a complete sequence (then the sum of all the durations)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

The presence of a fixed horizon allows to introduce a further metric taken from [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):

$$fldt = \sum_{i=1}^{n} \sum_{j=1 \wedge j \neq i}^{n} \frac{slack(a_i, a_j)}{H \times n \times (n-1)} \times 100 \tag{3.6}$$

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.[9] This metric characterizes the *fluidity* of a solution, i.e., the ability to use flexibility to absorb temporal variation in the execution of activities. Furthermore it considers that a temporal variation concerning an activity is absorbed by the temporal flexibility of the solution instead of generating deleterious domino effects (the higher the value of $fldt$, the lower the risk, i.e., the higher the probability of localized changes).

Example 3.4 *Figure 3.6 presents two different partial order schedules for two different problems. Both have the same activities while they have a different set of constraints: only precedence constraints in the latter, also time lags in the former. Also in this case, the two $\mathcal{POS}$s having "qualitatively" the same relations among*



(a) (b)

Figure 3.6: Two different $\mathcal{POS}$s for two different problems. While they have the same $flex$ value they are different with regard to $fldt$.

the activities have the same flexibility.

Let us consider the fluidity metric instead. We use the same horizon value $H = 10$ for both the situations (this is computed for the $\mathcal{POS}$ in Fig. 3.6(a)). It is possible to see that the two $\mathcal{POS}$s have different values of the fluidity metric $fldt$. In fact we obtain the following values $fldt(a) = \frac{1}{6}$ and $fldt(b) = \frac{1}{5}$.

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

The example above has proved the necessity of integrating the fluidity metric with one that is able to consider also the kind of relation between two activities.

The two metrics introduced above give an evaluation correlated with the number of feasible solutions *contained* in a $\mathcal{POS}$. An other aspect that is important to quantify is the stability of the solutions found. For this reason we define a further metric named disruptibility, $dsrp$; it measures the impact of a temporal change over the set of activities; we consider executions where only one unexpected event at a time can occur. The metric is defined as it follows:

$$dsrp = \frac{1}{n}\sum_{i=1}^{n}\frac{slack_{a_i}}{num_{changes}(a_i, slack_{a_i})} \tag{3.7}$$

where the function $num_{changes}(x, y)$ returns the number of activities that move from their original start time when the activity x is delayed by y units. In this case we always consider that the activity is moved to its latest start time, $slack_{a_i}$, in other words we evaluate the worst case situation.

In the next chapters the measures presented above are used to evaluate the quality of the solutions found and, then, the efficiency of the methods used. In order to produce an evaluation of the three criteria $fldt$, $flex$ and $dsrp$ that is independent from the problem dimension, we present the normalization of 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 initial network that represents the temporal aspect of the problem. In fact, for each metric the addition of precedence constraints between activities that are necessary to establish a resource-consistent solution can only reduce the initial value $\mu(\mathcal{P})$. Then the normalized value for the solution S of the problem $\mathcal{P}$ will have the following form:

$$|\mu(S)| = \frac{\mu(S)}{\mu(\mathcal{P})} \tag{3.8}$$

It is worth noting that the higher $|flex|$, $|fldt|$, and $|dsrp|$ the better.

3.6 Conclusions

In this chapter the crucial aspect of defining temporal flexible solutions has been tackled. The proposed solution lies in the introduction of a partial ordering among the activities which represents a set of feasible schedules. The model introduced has been named *Partial Order Schedule*, or $\mathcal{POS}$. As described in definition 3.1 a $\mathcal{POS}$ consists in 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.

Therefore as a scheduling problem consists itself of a set of partially ordered activities, generating a $\mathcal{POS}$ requires to post further temporal constraints on the initial

problem to guarantee that, once obtained, no further (temporal) constraint is able to produce a resource conflict. In other words, the solving process to obtain a partial order schedule, will return a new "problem" in which only the temporal aspect is presented while the combinatorial one has been "solved".

This aspect represents a significant point. In fact, any external change or unforeseen event that can be modeled as a temporal change in the problem, can be faced promptly thanks to the characteristics of the $\mathcal{POS}$. In general such a temporal flexibility can also be exploited in the face of different changes that stem from scheduling uncertainties. In general a $\mathcal{POS}$ retains a set of characteristics.

- It is a way to represent implicitly a set of different solutions.
- It provides a structure that allows a fast recovery.
- The minimality of repair actions avoids unnecessary domino effects preserving both the stability and the flexibility of the solution.

One important open question, though, is how to generate flexible schedules with good robustness properties. In the next chapters we investigate two different methods to obtain $\mathcal{POS}$s. The first, shown in Chapter 5, is an iterative repair method in which the set of all possible temporal solutions of the problem is considered at each stage of the solving process, pruning heuristically some of these alternatives until a $\mathcal{POS}$ is obtained. The second, detailed in Chapter 6, is a less intuitive approach in which partial order schedules are built on the basis of fixed-time solutions.

Chapter 4

Constraint-based Scheduling

In this chapter a generic framework to implement $\mathcal{POS}$ generation methods is described. This framework is based on the *Constraint Satisfaction Problem paradigm, or CSP*. A CSP consists of a network of constraints defined over a set of variables where a solution is an assignment to the variables that satisfies all the constraints. Effective methodologies based on this paradigm, can be obtained for both modeling the problem knowledge and guiding the search to a solution. Constraints do not simply represent the problem but play also an important role in the solving process narrowing the space of possible solutions. Constraint satisfaction and propagation rules are successfully used to model, solve and reason about many classes of problems such as scheduling, temporal reasoning, resource allocation, network optimization and graphical interfaces.

In particular the CSP approach has been proved to be efficient to model and solve complex scheduling problems (see for instance [Fox, 1990; Sadeh, 1991; Smith, 1994a; Beck *et al.*, 1998; Baptiste *et al.*, 2001; Cesta *et al.*, 2002]). A Constraint-based approach offers a solution to the problem of generating fixed-time schedules. The use of variables and constraints provides representational flexibility and reasoning power. For example, variables can represent the start and the end times of an activity, and these variables can be constrained in arbitrary ways. Along this chapter and in the rest of the thesis we exploit CSPs to face the problem of generating partial order schedules.

The chapter starts by introducing the concept of constraint satisfaction problem giving an high level description of the different aspects involved. Then, the CSP approach to scheduling problems is described. This is known in the literature as Constraint-based, or Constraint directed, Scheduling. Therefore, the general framework will be described. As mentioned before this will be the base on which different methods to produce $\mathcal{POS}$s, will be implemented.

4.1 Constraint Satisfaction Problem

A *constraint satisfaction problem*, CSP, consists of a finite set of variables, each associated with a domain of values, and a set of constraints that define the relation between the values that the variables can assume. More precisely, a CSP can be defined as a tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ where:

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is a set of n variables,
- $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ is the set of corresponding domains for any variable, that is, $v_1 \in D_1$, $v_2 \in D_2$ and $v_n \in D_n$,
- $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, is a set of m constraints, $c_k(v_1, v_2, \ldots, v_n)$, that are predicates defined on the Cartesian product of the variable domains, $D_1 \times D_2 \times \ldots \times D_n$.

A *solution* is a value assignment to each variable, from its domain,

$$\{\lambda_1, \lambda_2, \ldots, \lambda_n\} \in D_1 \times D_2 \times \ldots \times D_n$$

such that the set of constraints is satisfied. Constraint processing tasks include not only the satisfaction task, but also constraint optimization problems (COP or CSOP). In this case the solutions are not considered as equivalent, but a preference among the solutions is expressed. This is done by using an objective function (or cost function) that evaluates any single feasible assignment. Therefore, the goal will be to find the best solution (to optimize the objective function).

A fundamental aspect of constraint satisfaction problems is that an instance of a CSP $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ can be conceptualized as a constraint graph, $G = \{V, E\}$. For every variable $v_i \in \mathcal{V}$, there is a corresponding node in V. For every set of variables connected by a constraint $c_j \in \mathcal{C}$, there is a corresponding hyper-edge in E[1]. In the particular case in which only binary constraints (each constraint involves at most two variables) are defined the hyperedges become simply edges. A well known example of binary CSP is, for example, the Simple Temporal Network, or STP, [Dechter *et al.*, 1991].

Constraint Programming, or CP, is an approach to solving combinatorial, optimization, problems based on the CSP representation. In this framework the search for a solution to a CSP can be viewed as modifying the associated constraint graph by addition and removal of constraints. Thus the constraint graph is an evolving representation of the search state, where a solution is a state with a single value remaining in the domain of each variable, and all constraints are satisfied. Let us consider example 4.1 to show a constraint programming approach to the well known map coloring problem.

[1]We can observe that the formalism introduced in [Roy and Sussman, 1964] can be seen as a first attempt to model a scheduling problem like a CSP instance. In this work the authors describe a graph based formalism to represent a job-shop scheduling problem. This is clearly a CSP instance.

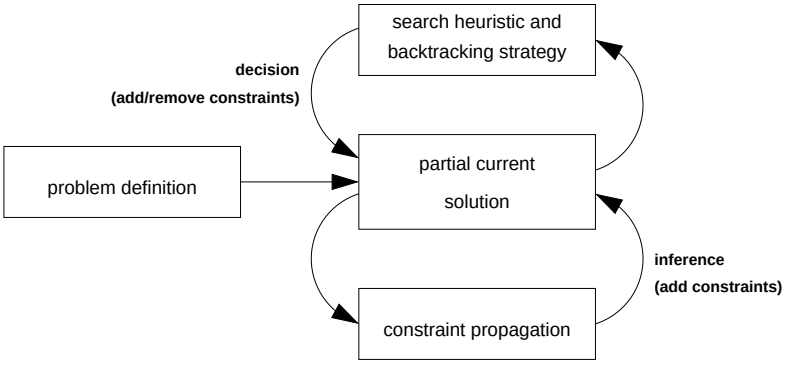


Figure 4.1: Constraint Programming at glance.

Example 4.1 *In the map coloring problem it is necessary to color each region of the map without assigning to any two adjacent regions the same color. This problem can be model by a (binary) CSP. Figure 4.2 shows an example: each region (four in the example) is represented by a variable, or node of the graph. The domain of each variable is the given set of colors (i.e., red, green, and yellow). For any pair of adjacent regions, there is a binary constraint, or edge in the graph.*

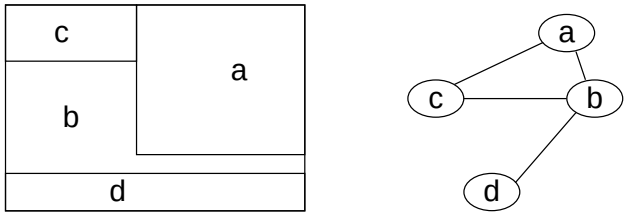


Figure 4.2: Map coloring problem.

Now we can add a unary constraint that assigns $a = green$*. Based on this constraint it is possible to infer the following changes of the domain of the remaining variable (regions):* $b \in [red, yellow]$*,* $c \in [red, yellow]$ *and* $d \in [red, yellow, green]$*. At this stage, as soon as either b or c will be assigned, the presence of the constraints it will infer the final assignment. For instance a feasible assignment is* $b = red$*,* $c = yellow$*, and* $d = green$*.*

Constraint Programming is based on the combination of sophisticated search technologies and constraint propagation. This combination is represented in Fig. 4.1 previously introduced in [Baptiste, 1998]. Constraint propagation consists of using constraints actively to prune the search space. Different filtering algorithms have been defined for different kinds of constraints. Their aim is to reduce the domains of variables involved in the constraints by removing the values that cannot be part of any feasible solution. This algorithm is invoked any time a domain of some variable is changed (both for search decision and constraint propagation) to propagate such a change over all the variables of the problem. As described in [Dechter and Rossi, 2002], "in general, constraint satisfaction tasks, like finding one or all solutions or the best solution, are computationally intractable, NP-hard". For this reason the constraint propagation process cannot be complete, that is, some infeasible values may still sit in the domains of the variables and thus decisions are necessary to find a complete feasible valuation of the variables. In general, propagation is aimed at achieving local consistency among subsets of variables.

With respect to the last point the concept of *k-consistency* is worth recalling: it guarantees that any locally consistent instantiation of $k-1$ variables is extensible to any k^{th} variable. Unfortunately, enforcing $k-$consistency can be accomplished in time and space exponential in k: if the problem has n variables, the cost of achieving $k-$consistency is $O(n^k)$. Therefore, it is clear that to achieve a global consistency (and prune then all the unfeasible solutions) it is necessary to assure $k-$consistency for any value $k = 1, \ldots, n$. Two tractable concepts of consistency are those defined for $k = 2$ and $k = 3$: respectively, arc and path consistency.

For the implementation of our framework, we have considered the Constraint Satisfaction Problem formalism, CSP, since it has the generality we desire to model scheduling problems. In fact, CSPs have successfully been used for a wide range of problems [Montanari, 1974; Kumar, 1992; Tsang, 1993].

4.1.1 Partial Order Schedules: why a constraint-based approach?

The ability of constraints to capture arbitrary relations make them "natural" modeling paradigm for our purposes. Unlike linear or integer programming frameworks, constraints are not restricted to linear equality and inequalities. In fact, it is possible to easily represent both mathematical or logical formulae and arbitrary relations with constraints. Therefore, the constraint programming approach satisfies the necessity of being able to represent the different aspects of the problem and retain the capability of guiding the search exploiting the knowledge of the problem. This requires, first, the creation of a rich constraint representation to capture relevant features and, second, to represent them in a way suitable for search techniques to reason upon them.

As mentioned before, constraint programming – that is a framework to solve problems based on the CSP representation – satisfies the need to represent the differ-

ent techniques and retains the ability of guiding the search exploiting the knowledge of the problem. These aspects turn out to be effective to approach complex combinatorial problems like in the case of project scheduling.

But there is a further motivation for using a constraint programming approach to generate partial order schedules. This stems from the definition of Partial Order Schedule. In Sect. 3.4.1 a partial order schedule is in fact defined as a set of solutions that can be represented by a temporal 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 for a given scheduling problem represented by the graph $G_P(V_P, E_P)$. Therefore it is clear how to obtain a partial order schedule, by posting a set of constraints over the original problem (the set E_R). It is worth noting that those constraints will be precedence constraints, that is constraints which relate the execution between pairs of activities. Of course, these constraints will have to be selected intelligently in order to have a $\mathcal{POS}$. Therefore the need to post new constraints to obtain a $\mathcal{POS}$ matches perfectly with the constraint programming approach. Indeed the latter allows to deal directly with constraints and particularly with precedence constraints.

The ability of managing precedence constraints during the solving process represents the main reason for which we have spent this chapter describing the Constraint Satisfaction paradigm and, in particular, the Precedence Constraint Posting model. In fact, solutions generated in this way generally represent a set of feasible schedules (i.e., the sets of activity start times that remain consistent with posted sequencing constraints), as opposed to a single assignment of start times. In the following chapters we will describe several algorithms based on the PCP model, in which $\mathcal{POS}$s are produced by adding (removing) new precedence constraints.

4.2 Scheduling + CSP = Constraint-based Scheduling

Scheduling problems belong to the area of combinatorial optimization problems therefore they can be easily represented as constraint satisfaction problems. In fact, different approaches have been developed. The work of Constraint directed Scheduling of the 80's (see for example [Fox, 1990; Sadeh, 1991; Smith, 1994a]) has developed into Constraint based Scheduling approaches in the late 90's (see [Baptiste and Le Pape, 1995; Nuijten and Aarts, 1996; Beck *et al.*, 1998]). These approaches are based on the representation of a scheduling problem and the search for a solution to it by focusing upon the constraints in the problem.

Given that even simple scheduling problems like job-shop scheduling are NP-hard [Garey and Johnson, 1979], the search process typically depends on heuristic commitments, propagation and retraction. In complex scheduling problems the goal is not simply to meet due dates but also to satisfy complex constraints. Therefore, scheduling represents an important application for constraint directed search. Different constraint programming approaches have been developed in this direction, for

Algorithm 4.1 Constraint Posting Scheduling Procedure

Input: A problem $\mathcal{P}$

Output: A solution $\mathcal{S}$

```
S_0 ← P
loop
  if termination condition are met then
    S ← S_0
    EXIT
  if S_0 is not consistent then
    retract commitment(s)
    if no commitment to retract then
      FAILURE
  else
    make heuristic commitment O(i, j, r) on S_0
```

instance, the reader can refer to [Baptiste *et al.*, 2001] for a thorough analysis of different constraint based techniques for scheduling problems.

As mentioned above, the search for a solution to a CSP can be viewed as modifying the constraint graph $G = \{V, E\}$ by addition and removal of constraints, where the constraint graph is an evolving representation of the search state, and a solution is a state with a single value remaining in the domain of each variable, and all constraints are satisfied.

Research in constraint-based scheduling (see [Nuijten and Le Pape, 1998]) has typically formulated the problem as that of finding a consistent assignment of start times for each goal activity. Under this model, decision variables are time points that designate the start times of various activities and CSP search focuses on determinating a consistent assignment of start time values. For instance, the RCPSP/max problem (Sect. 3.3) can be straightforwardly formulated in this way. A variable v_i is introduced for each activity; hence V is the set of CSP variables. For each variable v_i a domain $D_i = [0; H - p_i]$ is assigned (where H is an upper bound on the scheduling horizon), specifying its possible start times. Two types of constraint combine to further restrict the values that may be assigned to the set V of variables: (1) binary constraints (involving pairs of variables) for representing the start-to-start temporal relations between activities; and (2) n-ary constraints to describe the capacity constraints that each resource imposes on all feasible schedules.

In contrast, we are pursuing a model to scheduling that operates with a problem formulation more akin to least-commitment planning frameworks: the *Precedence Constraint Posting*, PCP, model (see [Smith and Cheng, 1993; Cheng and Smith, 1994; Oddi and Smith, 1997; Cesta *et al.*, 2002]). This approach consists of post-

ing a sufficient set of additional precedence constraints; these constraints will relate pairs of activities that contend the same resources ensuring the feasibility with respect to time and resource-capacity constraints. The decision variables correspond alternatively to the various ordering decisions that need to be made between sets of activities that are competing for the same resources, and CSP search focuses on specifying (or posting) a consistent set of precedence constraints that eliminates any possibility of resource contention. One principal advantage of this sequencing approach is that it avoids over-commitment, as activities need not be anchored to specific start times during the search or in the final solution (this provides representational flexibility).

Algorithm 4.1 describes a high level procedure for a constraint programming approach to a scheduling problem in a PCP perspective. To formulate RCPSP/max under this model, a variable is defined for each set of activities that simultaneously require resource r_k with a total capacity requirement greater than c_k (i.e., each potential resource conflict). For each variable v_i, the domain $D_i = \{pc_{i,1}, \ldots\}$ consists of the set of precedence constraints that can be feasibly posted to eliminate the conflict. Operationally, the ongoing determination of the domain D_i for each ordering decision variable v_i requires the propagation of activity start times in an underlying time-point network. Thus, a precedence constraint posting model can be seen as a meta-CSP formulation, which utilizes a start-time assignment model as a ground-CSP representation.

In the next section we describe the details of the precedence constraint posting approach and, then, introduce the framework used to generate partial order schedules.

4.3 Precedence Constraint Posting

Precedence constraint posting approaches aim at synthesizing additional precedence constraints between pairs of activities for the purpose of pruning all inconsistent allocations of resources to activities. The general schema of these approaches is provided in Fig 4.3. The approach consists in representing, analyzing, and solving different aspects of the problem into two separate layers. In the former the temporal aspects of the scheduling problem like activities duration, constraints between pair of activities, due dates, release time, etc., are considered. The second layer, instead, represents and analyzes the resource aspects of the problem[2]. Let us now explain the details of the two layers.

Time layer. The temporal aspects of the scheduling problems are represented through an STP (simple temporal problem) network [Dechter *et al.*, 1991]. This is a temporal graph in which the set of nodes represents a set of temporal variables named

[2] A similar distinction between temporal and resource aspects of the scheduling problem is pursued in [El Sakkout and Wallace, 2000].

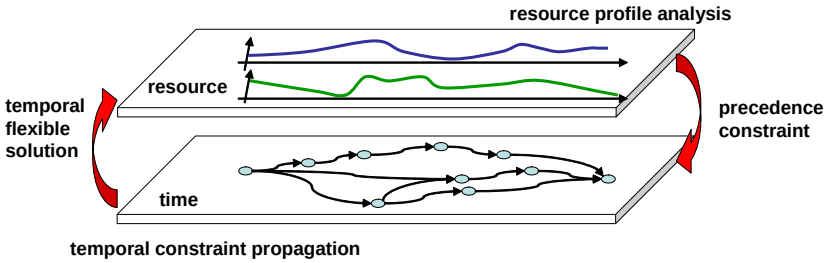
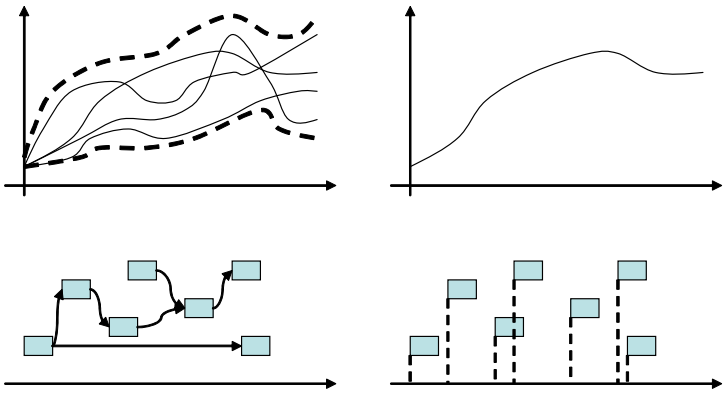


Figure 4.3: Precedence Constraint Posting Schema.

time-points, tp_i, while temporal constraints, of the form $tp_i - tp_j \leq d_{ij}$, define the distances among them. Each time point has initially a domain of possible values equals to $[0, H]$ where H is the horizon of the problem (generally H can be infinite). The problem is represented associating to each activity a pair of time points which represent, respectively, the start and the end time of the activity. Therefore a temporal constraint may be defined between a pair of time points that can "belong" or not to the same activity. In the latter case (when they do not belong to the same activity) the temporal constraints represent constraints between two activities of the problem, while if the two time-points belong to the same activity the temporal constraints represent the duration, or processing time, of the activities. By propagating the temporal constraints it is possible to bound the domains of each time-point, $tp_i \in [lb_i, ub_i]$. In the case of empty domains for one ore more time-points the temporal graph does not admit any solution. In [Dechter *et al.*, 1991] has been proved that it is possible to completely propagate the whole set of temporal constraints in polynomial time, $O(n^3)$, and, moreover, a solution can be obtained selecting for each time-point its lower bound value, $tp_i = lb_i$ (this solution is named *earliest start-time solution*).

The temporal layer then, given the temporal aspects of a scheduling problem, provides, in polynomial time (using constraint propagation) a set of solutions defined by a temporal graph. This result is taken as input in the second layer. In fact, at this stage we have a set of temporal solutions (time feasible) that should be proved to be also resource feasible.

Resource layer. This layer takes into account the other aspect of the scheduling problem, namely resources. The problem is that there are constraints to resource utilization (i.e., capacity). Resources can be binary or multi capacitive, and reusable or consumable. As described above the input of this layer in the PCP approach is a temporally flexible solution – a set of temporal solutions (see also Fig 4.3). Like in the previous layer to reduce the search space it is possible to use constraint propagation (i.e. resource propagation). Even though there are different methodologies described in the literature, see [Nuijten and Aarts, 1996; Laborie, 2003]), these are not suffi-



(a) Bounds of the resource utilization for the set of solutions defined by a temporal graph

(b) Resource utilization of a single temporal solution

Figure 4.4: Two different ways to consider the resource utilization

cient in general. In fact these are not complete, that is, they are not able to prune all the inconsistent temporal solutions. Therefore, it is necessary to use a method which allows to decide among the possible alternatives. For this reason the method uses a *Resource Profile* to analyze resource usage over time and detect periods of resource conflict or contention peaks. The method proceeds posting further constraints to solve some of the detected peaks. These new constraints are propagated in the underlying layer to check the temporal consistency. Then the time layer provides a new temporally flexible solution that is analyzed using the resource profiles. The search stops when either the temporal graph becomes inconsistent or the resource profiles are consistent with the resource capacities.

The next section is dedicated to describing heuristics for the analysis of the resource profile and the synthesis of new constraints. However, the main issue that needs to be explored is how to compute these profiles. In fact, the input temporal graph represents a set of solutions, possibly infinite, and to consider all the possible combinations is impossible in practice. A possible affordable alternative consists in computing bounds for the resource utilization. Examples of bounds can be found in [Drabble and Tate, 1994; Cesta and Stella, 1997; Laborie, 2003; Muscettola, 2002]. It is worth noting that considering the resource bounds as resource profiles assures to generate a partial order schedule at the end of the search process. In fact, this stops when the resource profile is consistent with respect to the capacity constraints, that means that all the temporal solutions represented by the temporal graph are also resource feasible (see Definition 3.1).

A different approach to deal with resources consists in focusing the attention on

a specific temporal solution and its resource utilization. Even though the precedence constraint posting method produce a temporal graph when the resource utilization is consistent with resource capacity, this graph, in general, is not a $\mathcal{POS}$. Indeed, this process only assures that the final graph contains at least one resource feasible solution (the one for which the resource utilization is considered); some of the temporal solutions may not be resource feasible. Thus it is necessary to use a method which is capable of transforming the resulting temporal graph into a partial order schedule.

Figure 4.4 summarizes the two alternative resource profiles. In the first case resource bounds are used to consider all the temporal solutions and their associated resource utilization (Fig. 4.4(a)). Alternatively, only one temporal solution of the set is considered. This allows to reason about one precise resource profile but, in fact, produces temporal graphs that are not in general $\mathcal{POS}$s.

In the next chapter we will see how different ways of computing and using resource profiles lead to different PCP-like algorithms. The next section, as anticipated, describes, instead, the very framework used in the implementation of methods aiming at generating $\mathcal{POS}$s.

4.3.1 The Core Constraint-based Scheduling Framework

The core of the implemented framework is based on the greedy procedure described in Algorithm 4.2. Within this framework, a solution is generated by progressively detecting time periods where resource demand is higher than resource capacity and posting sequencing constraints between competing activities to reduce demand and eliminate capacity conflicts. As explained above, after the current situation is initialized with the input problem, $S_0 \leftarrow \mathcal{P}$, the procedure builds an estimate of the required resource profile according to the current temporal precedences in the network. This analysis can highlight contention peaks, where resource needs are greater then resource availability (Select-Conflict-Set(S_0)). If the set of conflict C_s is not empty then new constraints are synthesized (Select-Leveling-Constraint(C_s)) and posted on the current situation. The search proceeds until either the temporal graph becomes inconsistent or a solution is found.

We proceed now by introducing the core components that need to be explained to complete the description of the approach. The first issue consists in how to identify activities which are in a conflicting situation. This allows to know the points in the current situation that need to be solved. The second point is represented by both heuristics and methods used to respectively select and solve one of the conflicts that have been singled out.

How to identify conflicts. The starting point in identifying the possible conflicts in a situation is to compute the possible contention peaks or resource violations. A couple of definitions are necessary before proceeding. First, a *contention peak* is

Algorithm 4.2 greedyPCP($\mathcal{P}$)

Input: a problem $\mathcal{P}$

Output: a solution $\mathcal{S}$

```
S_0 ← P
if Exists an unresolvable conflict in S_0 then
    FAILURE
else
    C_s ← Select-Conflict-Set(S_0)
    if C_s = ∅ then
        S ← S_0
    else
        (a_i{before}a_j) ← Select-Leveling-Constraint(C_s)
        S_0 ← S_0 ∪ {a_i{before}a_j}
        S ← greedyPCP(S_0)
return S
```

a set of activities whose simultaneous execution exceeds the resource capacity. A contention peak designates a conflict of a certain size (corresponding to the number of activities in the peak). Second, a *conflict* is a pair of activities $\langle a_i, a_j \rangle$ belonging to the same contention peak. The function Select-Conflict-Set(S_0) of Figure 4.2 collects all the peaks in the current schedule, ranks them, picks the more critical one and then selects a conflict from this last peak. The conflict is solved by ordering the conflicting activities with a new precedence constraint, $a_i \prec a_j$.

A first way to extract a conflict from a peak is the *pairwise selection* [Smith and Cheng, 1993]. This consists of collecting any competing activity pairs associated with each peak. The myopic consideration on any pair of activities in a peak can, however, lead to an over commitment. For example, consider a resource r_j with capacity $max_j = 4$ and three activities a_1, a_2 and a_3 competing for this resource. Assume that each activity requires respectively 1, 2 and 3 units of the resource. Taking into account all possible pairs of activities will lead to consideration of the pair $\langle a_1, a_2 \rangle$. But the sequencing of this pair will not resolve the conflict because the combined capacity requirement does not exceed the capacity.

An enhanced conflict selection procedure which can avoid this problem is based on the identification of *Minimal Critical Sets* [Laborie and Ghallab, 1995] inside each contention peak. A *Minimal Critical Set*, MCS, is *a conflict such that no proper subset of activities contained in the* MCS *is itself a conflict*. The important advantage of isolating MCSs is that a single precedence relation between any pair of activities in the MCS eliminates the resource conflict. Let us consider again the previous example: in this case the only MCS is $\{a_2, a_3\}$ and both the precedence constraints $a_2 \prec a_3$

and $a_3 \prec a_2$ solve the peak. Application of this method can be seen generally as a filtering step. Indeed, it extracts from each contention peak those subsets that are necessary to solve.

MCS analysis has been used in [Laborie and Ghallab, 1995], where MCSs are seen as particular cliques that are collected via systematic search of an activity "intersection graph" (which is constructed starting from the temporal information). The unfortunate drawback of this approach is the exponential nature[3] of the intersection graph search, which prohibits the use of this basic approach on scheduling problems of any interesting size. In [Cesta *et al.*, 1998], it is shown that much of the advantage of this type of global conflict analysis can be retained by using an approximate procedure for computing MCSs. For these reasons here we import the following sampling strategies:

Linear sampling. A queue Q is used to select an MCS from a contention peak P. Activities a_i are considered sequentially and inserted in Q until the sum of the resource requirement is greater than the resource availability. Then the set Q is saved in a list of MCS and the first element in Q is removed. The previous steps are iterated until there are no more activities. The complexity of this method is $O(p)$, where p is the size of the peak;

Quadratic sampling. This is an extension of the previous schema in which the second step is expanded as follows. Once the correct MCS has been collected, instead of removing the first element from Q a forward search through the remaining activities is performed to collect all MCS that can be obtained by dropping the last item placed in Q and substituting it with single subsequent activities until an MCS is composed. Under this scheme, a larger subset of MCSs are selected using a procedure of complexity $O(p^2)$.

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

[3] In [Laborie and Ghallab, 1995] minimal critical sets are detected as maximum cliques on a specific representation of temporal constraints, the *possible intersection graph* (PIG). For any resource r_k an associated PIG is computed: this consists of graph $G_k(U, E)$ where U is the set of activities that require the use of resource r_k (i.e., $U = \{a_i | req_{ik} \neq 0\}$), and E is a set of edges between the pair $(a_i, a_j) \in U \times U$ such that they may possibly overlap (i.e., $a_i \not\prec a_j$ and $a_j \not\prec a_i$). PIGs are a special kind of weakly triangulated graphs that are themselves a sub-class of perfect graph.

In [Hayward *et al.*, 1989] it has been proved that the maximum cliques for a weakly triangulated graphs can be found in polynomial time. Nevertheless this algorithm is very inefficient in average and furthermore it would give only some MCS and not necessarily the most interesting.

Select and solve conflicts. Independently of whether the conflict selection is performed directly from activity pairs or from sampled MCSs, a single conflict is selected for resolution according to the "most constrained first" principle. Given a selected pair of conflicting activities, the order between them will be chosen according to a "least constraining" principle. As it is underlined in [Smith and Cheng, 1993], given a pair $\langle a_i, a_j \rangle$ of activities in a given contention peak, four possible conditions can be held between the two activities according to the maximum distance, $d()$, between two events:

condition 1 : $d(e_{a_i}, s_{a_j}) < 0 \wedge d(e_{a_j}, s_{a_i}) < 0$. In this case there is no way to order the activities. This is identified as a *pairwise unresolvable conflict*.

condition 2 : $d(e_{a_i}, s_{a_j}) < 0 \wedge d(e_{a_j}, s_{a_i}) \geq 0 \wedge d(s_{a_i}, e_{a_j}) > 0$. There is only one feasible ordering the two activities $a_j\{before\}a_i$.

condition 3 : $d(e_{a_i}, s_{a_j}) \geq 0 \wedge d(e_{a_j}, s_{a_i}) < 0 \wedge d(s_{a_j}, e_{a_i}) > 0$. Like the previous one this is also a *pairwise uniquely resolvable conflict*. In this case the relation is $a_i\{before\}a_j$.

condition 4 : $d(e_{a_i}, s_{a_j}) \geq 0 \wedge d(e_{a_j}, s_{a_i}) \geq 0$. In this case we have a *pairwise resolvable conflict*. Both orderings $a_i\{before\}a_j$ and $a_j\{before\}a_i$ are feasible and a choice is needed.

This procedure returns the ordering constraint that leaves the most temporal flexibility: $a_i\{before\}a_j$ if $d(e_{a_i}, s_{a_j}) > d(e_{a_j}, s_{a_i})$ and $a_j\{before\}a_i$ otherwise.

As the reader can see, decisions are taken according to a least commitment principle, trying to retain the maximum amount of temporal flexibility. For that reason the values of the distances $d(e_{a_i}, s_{a_j})$ and $d(e_{a_j}, s_{a_i})$ have a key role. The basic idea is to resolve the conflict that is the most "dangerous" and solve it with a commitment as small as possible. More specifically, the following heuristics are assumed:

Ranking conflicts: for evaluating the contention peaks we have used the heuristic estimator K() described in [Laborie and Ghallab, 1995]. Given a contention peak and a set of possible ordering constraints $\{oc_1, \ldots, oc_k\}$ which can be posted between pairs of the activities the estimator $K()$ is defined:

$$\frac{1}{K()} = \sum_{i=1}^{k} \frac{1}{1 + commit(oc_i) - commit(oc_{min})} \tag{4.1}$$

where $commit(oc_i)$ estimates the loss in temporal flexibility of posting the ordering constraint $oc_i = \{a_i \prec a_j\}$

$$commit(oc_i) = \frac{\min(d_{ij}^{max}, 0)\min(d_{ij}^{min}, 0)}{d_{ij}^{max} - d_{ij}^{min}} \tag{4.2}$$

where d_{ij}^{min} and d_{ij}^{max} are, respectively, the minimum and maximum distance between a_i and a_j. The heuristic estimator $K()$ chooses the contention peak with highest value. The conflict resolution heuristic simply chooses oc_{min}. A conflict is unsolvable if no pair of activities in the conflict can be ordered. Basically, $K()$ will measure how close a given conflict is to being unsolvable.

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

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

4.4 Summary

This chapter describes the framework within which we shall introduce the methods that are the objects of the remainder of this work. This is based on a well known paradigm: Constraint Satisfaction Problem (CSP). In the literature it is possible to find different examples that exploit the effectiveness of this model to tackle complex scheduling problems.

More precisely, we use a particular CSP formulation of a scheduling problem: the Precedence Constraint Posting model. This approach consists of posting the set of additional precedence constraints that are needed among the sets of activities that are competing for the same resources. One principal advantage of this sequencing approach is that it avoids over-commitment, as activities need not be anchored to specific start times during the search or in the final solution.

The model is based on the separation of the time and the resource aspects (see Fig. 4.3). At a first layer time aspects of the problem are considered producing a temporally flexible solution. This solution is analyzed in a different layer to check resource feasibility. This analysis is based on the construction of resource profiles; as it has been shown, these can be associated to either a set of solutions or a single one (this distinction creates in the next chapters two different alternatives to produce $\mathcal{POS}$s). Then different techniques have been introduced for analyzing the resource profile, identifying conflicts (i.e., subsets of conflicting activities), selecting, and then solving, one of them.

Chapter 5

A Least Commitment Approach

This chapter describes an approach to generate Partial Order Schedules, that is based on the well-known paradigm named *Least Commitment*. The basic idea behind least commitment consists of reducing as much as possible the commitment implied by a decision during the solving process. There are two fundamental aspects related to this strategy: postponing all not necessary decisions as much as the search procedure allows it and choosing the least constraining for the current solution process.

In particular, the described method can be viewed as an iterative repair method in which the set of all possible temporal solutions of the problem is considered at each stage of the solving process. This set is then analyzed computing resource usage bounds (i.e. resource envelope [Muscettola, 2002]) and based on this analysis some of the temporal solutions are heuristically pruned. The process continues until a $\mathcal{POS}$ is produced, i.e., the resource usage bounds are consistent with the resource capacity.

5.1 Introduction

In the perspective of a "pure" least commitment approach a scheduler should consist in carrying out a refinement search that incrementally restricts a partial solution (the possible temporal solutions $\tau \in S_T$) with resource conflicts until a set of solutions (a $\mathcal{POS}$ in our case) is identified. For this reason, in the construction of a baseline schedule it is necessary to have a kind of lens to analyze and manage the current situation. For instance, the PCP framework described in Sect. 4.3 represents the current solution by using a *Time Network* or *Simple Temporal Problem*, STP. In this representation each activity is a node while the edges represent all the temporal constraints which the activities have to respect. Moreover to keep trace of the resource usage, a resource profile is built for each resource in order to check possible contention peaks (or peaks). In case a peak exists then, in order to remove it, the

search process proceeds by synthesizing a new precedence constraint between a pair of activities. Applying the least commitment approach requires analyzing the three following aspects:

- the way in which contention peaks are detected, i.e. which is the algorithm used to identify peaks;
- the way in which a conflict – a set of activities which are responsible for the peak – is selected;
- the way in which the selected conflict is solved.

It is worth noting that an important limitation to the application of a least commitment approach so far, has been the lack of accurate bounds for the resource profile. The recent introduction of the concept of resource envelope [Muscettola, 2002] can contribute to the effectiveness of this type of approach. This technique in fact allows an exact computation of the *tightest possible resource-level bound for a flexible plan*. The Resource Envelope can be defined as follows:

Definition 5.1 (Resource Envelope) *Let S_T be the set of temporal solutions. For each resource r_j we define the Resource Envelope in terms of two functions:*

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

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

The integration of the envelope computation into a PCP algorithm is quite natural. In fact, it is used to restrict resource profile bounds, in accordance with the current temporal constraints in the underlying "simple temporal network". In [Muscettola, 2002] it is proved that it is possible to find the Resource Envelope through a polynomial algorithm. The advantage of using the resource envelope is that all possible temporal allocations are taken into account during the solving process.

This chapter describes the first scheduling algorithm that uses this technical result. Our goal is twofold: (a) to obtain a method to compute partial order schedules; (b) to analyze the effective improvement that the integration of the resource envelope into a precedence constraint posting framework can give. This chapter starts recalling the concept of resource envelope. Sect. 5.3 presents a set of rules to incrementally compute the envelopes that are very useful to speed up the resource envelope computation, after which Sect. 5.4 introduces a first greedy implementation of a resource envelope based scheduler. Even though it is a rather simple algorithm, it has been important to understand critical points in the implementation of the resource envelope solver and, above all, to identify the drawbacks of this approach and to lead possible ways to solve them. Finally Sect. 5.5 presents an improved algorithm that integrates resource propagation into a more intensive analysis of the search space.

5.2 Compute Resource Bounds

To introduce the concept of resource envelope we need a slightly different notation to represent scheduling problems. It is based on events that change the resource availability rather than the whole activity.

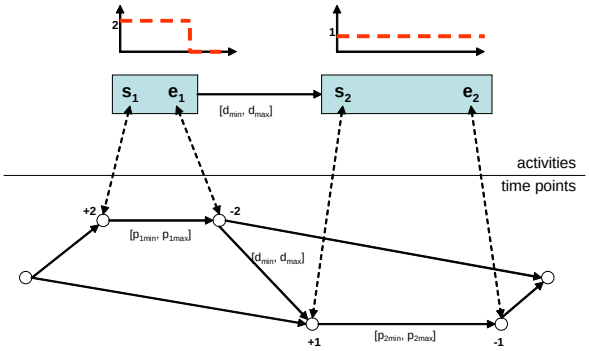


Figure 5.1: Events-based representation.

Event-based Representation. Figure 5.1 shows an example of the event-based representation for a two activities, one resource, problem. The left hand activity requires two resource units to be executed, then at the end of the execution it releases them. The second activity, instead needs one resource unit to be executed but this is not released (consumer activity). The events based representation consists in associating at each activity time point, the start and the end time, the resource usage event. For instance, in the case of the first activity you have two time points with two associated events: $ru = +2$ at the start time and $ru = -2$ at end time. For the second activity you have respectively: $ru = +1$ and $ru = 0$. It is worth noting that between two time points (or events) it is possible to have constraints which have a different semantics: (1) precedence constraints when the time points belong to two different activities and (2) activity duration when the time points belong to the same activity. We recall that in the case of the problem we refer to, RCPSP/max, there are only the first type of activities, thus for each activity a_i which requires req_{ij} units of resource r_j, there is an event at the start time with associated resource usage $ru = req_{ij}$ and and event with an associated resource usage equals to $ru = -req_{ij}$ at the end time. To refer to an event, in the remaining of the section we use without distinction both the term event and time point.

Partition. Once represented the problem with the event based notation we can use it to describe how the resource envelope can be computed. For the computation of the

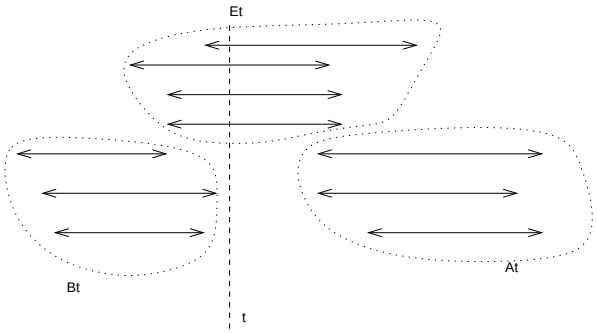


Figure 5.2: The partition into B_t, E_t and A_t.

tightest resource bound, i.e. the maximum (minimum) value of the resource level at any instant t, the following partition of the set of time points (or events) is considered:

- B_t: the set of events t_i s.t. $lst(t_i) \leq t$, i.e., the events which must have occurred by the instant t;
- E_t: the set of events t_i s.t. $est(t_i) \leq t < lst(t_i)$, i.e., the events which can occurr at time t;
- A_t: the set of events t_i s.t. $est(t_i) > t$, i.e., the events which have to occur after time t.

Based on this partition, the contribution of any event t_i to the maximum (minimum) resource value can be computed according to which of the three sets the event t_i belongs to. In fact, since the events in B_t are those which happen before or at time t, they all contribute - with the associated resource usage ru_{ij} - to the value of the resource profile of r_j in the instant t. By the same argument we can exclude from this computation the events in A_t, as they happen after t. Thus is evident that the maximum (minimum) resource value depends on the maximum (minimum) contribution that the events in E_t may give. For instance, the maximum resource usage at time t is:

$$L_j^{max}(t) = \sum_{t_i \in B_t} ru_{ij} + \sum_{P_{max}(E_t)} ru_{ij} \tag{5.1}$$

where $P_{max}(E_t)$ is a subset of E_t which gives the maximum contribution among any combination of elements in E_t can give.

A trivial and, unfortunately, expensive approach to extract the subset P_{max} from the set of pending events E_t consists in enumerating all the possible combinations. Indeed this approach might require an exponential CPU time to compute the resource bounds which makes this approach unrealistic. A method to overcome this problem

has been introduced in [Muscettola, 2002]. There the author describes a polynomial algorithm to compute the set P_{max} (P_{min}). The following section describes this concept.

5.2.1 Resource Envelopes

In [Muscettola, 2002] the author proves that to find subset of E_t for computing the upper (lower) bound, it is possible to avoid enumerating all the possible combinations of events in E_t. Muscettola shows that a polynomial algorithm can be found through a reduction of the problem to the Max-Flow problem [Ford and Fulkerson, 1962], a well-known tractable problem. The effectiveness of the reduction is due to the fact that it is possible to exploit the relations among the set of events and to consider only a subset of all the combinations.

The Max-Flow Problem. In the following we briefly review the theory behind the Max-Flow problem [Cormen *et al.*, 1990]. A flow network $G(V, E)$ is a directed graph where V is a set of nodes and E is a set of edges (u, v) with nonnegative capacity $c(u, v) \geq 0$. The flow network has two special nodes: a source s and a sink t. A flow in G is a real-valued function $f : V \times V \rightarrow \mathbb{R}$ that satisfies the following three properties:

- for all $u, v \in V$, $f(u, v) \leq c(u, v)$
- for all $u, v \in V$, $f(u, v) = -f(v, u)$
- for all $u \in V \cup \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$.

The value of a flow f into the graph G, is defined as

$$f = \sum_{v \in V} f(s, v),$$

that is the total flow out of the source. In the Max-Flow problem given a flow network G, the goal is to find a flow of maximum value from source to sink. A fundamental concept in flow theory is the *residual network*: a graph with an edge for any pair of nodes which has a positive residual capacity, i.e., the difference $c(u, v) - f(u, v)$ between the capacity and flow through the edge.

The Resource Envelope Computation. As we mentioned before the method computes the resource envelope values building a Max Flow problem. More precisely to find the set $P_{max}(E_t)$ a Max-Flow problem is constructed using the anti-precedence graph defined as follows:

Definition 5.2 *The anti-precedence graph for the resource R, $Aprec(R)$ is a graph with the same events and such that for any two events e_1 and e_2 with $|e_1 e_2| = 0$ there is a path from e_1 to e_2 in $Aprec(R)$.*

Given the anti-precedence graph the Max-Flow problem is built according to the following rules:

1. all events $t_i \in E_t$ are represented by nodes t_i of the flow problem. Two further nodes are added to represent the source σ and the sink τ.

2. if exists a precedence $t_i \prec t_j$ between the two events t_i and t_j in the anti-precedence graph, then the flow problem contains an edge with infinite capacity between the nodes associated with the two events;

3. for any producer event t_i, i.e. $ru_{ij} > 0$, there is an edge between the source σ and t_i with capacity ru_{ij};

4. for any consumer event t_i, i.e. $ru_{ij} < 0$, there is an edge between the node t_i and the sink τ with capacity $|ru_{ij}|$;

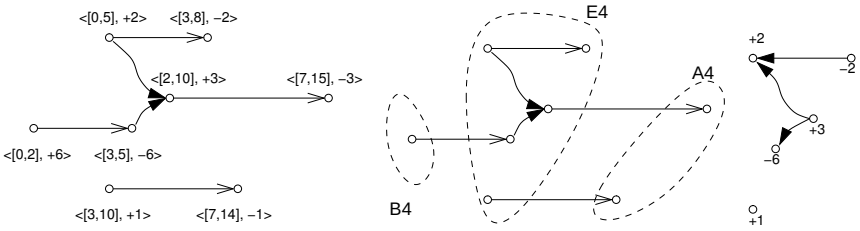
An example of flow problem construction is shown in Fig. 5.3. Let us consider the instant $t = 4$. For this time point, Fig. 5.3(a) and 5.3(b) represent respectively the scheduling problem and the associated partition described above. Figure 5.3(c) shows the anti-precedence graph associated to E_4. Based on this a flow network is built (Fig. 5.3(d)). It is possible to note that in the Max Flow problem obtained following the rules above, any maximum flow through the flow network matches all possible producer event with all possible consumer events according to the precedence constraints. In [Muscettola, 2002] it has been proved[1] that the set of events reachable from the source through the residual flow is equivalent to the set P_{max}. In the example you have that only one node is reachable form the source (see Fig. 5.3(f)). Then the set P_{max} contains only the time-points associated with these nodes which have a value equals to $+1$. Therefore, the resource envelope value for $t = 4$ is equals to

$$L_j^{max}(4) = \sum_{t_i \in B_4} ru_{ij} + \sum_{P_{max}(E_4)} ru_{ij} = +6 + 1 = 7$$

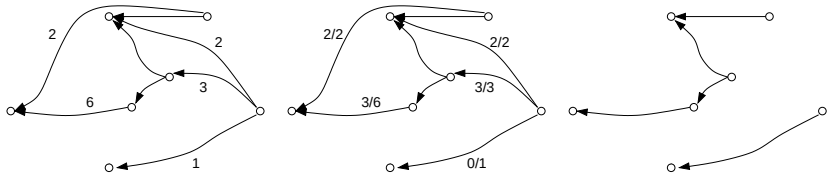
according to (5.1).

To summarize the algorithm consists in building a Max-Flow problem from the set of events belonging to E_t and, after the max flow is found, the subset $P_{max} \subseteq E_t$ ($P_{min} \subseteq E_t$), of events that gives the maximum (minimum) value of the resource level at the instant t, is computed performing a linear analysis of the residual graph. An important point is that it is not necessary to compute the resource-level envelope

[1] Theorem 1 in [Muscettola, 2002].



(a) Initial Problem (b) Partition into B_4, E_4 and A_4 (c) Anti-precedence graph of E_4



(d) Flow Network (e) A Maximum Flow solution (f) Residual Graph

Figure 5.3: Resource Envelope Computation: Example

in all possible instants t. Indeed, you only need to compute L_j^{max} at times when either B_t or E_t changes. For any time point t_i this can only happen either at its earliest time value $est(t_i) = -d(t_i, t_0)$ or at its latest time value $lst(t_i) = d(t_0, t_i)$. Thus the approach has a complexity of $O(nO(maxflow(n)))$ where n is the number of events in the problem and $O(maxflow(n))$ is the time complexity of a Max Flow problem with n elements. Depending on the solving method[2] the time complexity of the Max Flow problem can be $O(n^3)$, in case of the original augmenting path method proposed in [Ford and Fulkerson, 1962] or $O(n^2m)$, where m is the number of edges, in case of the augmenting path version of Edmond-Karp or in the *preflow-push* method [Goldberg and Tarjan, 1988]. Therefore computing the resource envelope has a complexity of $O(n^4)$ or $O(n^3m)$ with respect to the Max-Flow algorithm used.

On this last point, unfortunately, lies a critical aspect of the algorithm proposed by Muscettola. In the next section we analyze this point and we propose a set of rules to limit the CPU time requested to compute the resource envelope.

[2] A thorough treatment can be found for instance in [Ahuja *et al.*, 1993]

5.3 Boosting the Resource Envelope Computation

A potential drawback in using an envelope computation within a scheduling algorithm like the base PCP solver introduced in Sect. 4, is the computational burden of the Max-Flow computation. Despite being polynomial, the computational cost is significant and can become a limiting factor to scale up the addressed scheduling problem. In fact, they require a new computation of the resource envelope for each resource, each time a new constraint is posted.

In this section, we establish some properties for computing the envelope incrementally across points of discontinuity. Moreover since the incremental methods to compute P_{min} and P_{max} can be obtained from each other with obvious term substitutions, we only develop the method for P_{max}. Our goal is to avoid the computation of P_{max} using the maximum flow algorithm, or, at least, to reduce the number of times that the resolution of the associated Max-Flow problem is requested, and, in that case, identify the sub-network which determines the differences between two "successive" Max Flow instances. The basic idea is to exploit the knowledge which comes from the computation of the value $L_j^{max}(t-1)$, to compute the value of $L_j^{max}(t)$; that is, we are looking for the hypothesis under which is possible to compute the value:

$$\Delta L_j^{max}(t) = L_j^{max}(t) - L_j^{max}(t-1)$$

avoiding the use of the Max-Flow reduction.

To introduce the rules for an incremental computation of the resource envelope we first need to define the overall contribution of a time point t_i to the resource envelope value.

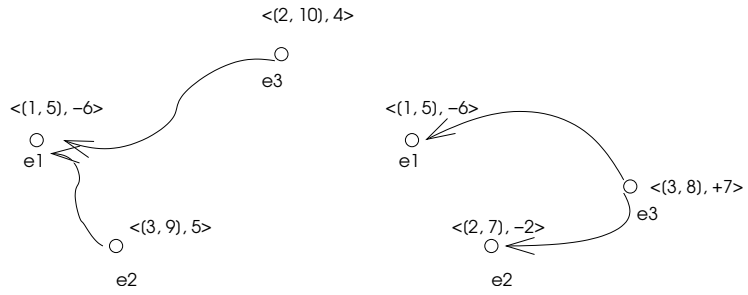
Definition 5.3 *Given a time point t_i and a resource r_j we define its overall contribution to be the value:*

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

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

Figure 5.4(a) shows an anti-precedence graph in which there are two production events and a consumption one. It is possible to note that the contribution of each production is negative, $\Sigma ru_{2j} = ru_{2j} + ru_{1j} = 5 + (-6) = -1$ and $\Sigma ru_{3j} = ru_{3j} + ru_{1j} = 4 + (-6) = -2$ while the set $\{e_1, e_2\}$ gives a positive contribution, $ru_{1j} + ru_{2j} + ru_{3j}$. This implies that only a conjunctive presence in P_{max} of the two events is allowed.

Now to find possible rules to compute the resource envelope incrementally, we consider the possible evolution of the three sets B_t, E_t and A_t of the partition, across points of discontinuity. It is possible to note the following cases:



(a) Each production has a negative contribution while their conjunction is positive

(b) The set E_t is empty for $t < 5$

Figure 5.4: Examples of the event contribution over the resource bounds

1. event t_i moves from E_t to B_{t+1}, that is $t_i \in E_t \cap B_{t+1}$;
2. event t_i which remains pending with the increment of time t, that is $t_i \in E_t \cap E_{t+1}$;
3. event t_i which becomes pending, that is $t_i \in A_t \cap E_{t+1}$.
4. event t_i moves from A_t to B_{t+1}, that is $t_i \in A_t \cap B_{t+1}$.

In the last case, it is possible to note that any $t_i \in A_t \cap B_{t+1}$ contributes to the value of $L_j^{max}(t)$ in the amount corresponding to its resource usage ru_{ij}. In the second case listed above, the following theorem allows us to restrict the set of time points for which P_{max} must be computed:

Theorem 5.1 *If there exists a time point $t_i \in E_t \cap E_{t+1}$ and $t_i \in P_{max}(E_t)$, then $t_i \in P_{max}(E_{t+1})$.*

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

From this theorem it follows that at each instant t it is sufficient to consider only the events in $E_t \backslash P_{max}(E_{t-1})$ to figure out which events have to be collected into $P_{max}(E_t)$. Indeed any time-point $t_i \in P_{max}(E_{t-1})$ still contributes to the value of $L_j^{max}(t)$ both if $t_i \in B_t$ and if $t_i \in E_t$. Furthermore, from the previous theorem, it is possible to prove the following corollary:

Corollary 5.1 *If* $E_{t+1} \backslash P_{max}(E_t) = E_t \backslash P_{max}(E_t)$ *then* $P_{max}(E_{t+1}) = P_{max}(E_t)$.

Now considering the subset $P_{max} \subseteq E_t$ we can rewrite the first point of the previous list as:

1.a events move from $P_{max}(E_t)$ to B_{t+1}

1.b events move from $E_t - P_{max}(E_t)$ to B_{t+1}

Since we know the value of $L_j^{max}(t)$ in the case (1) we compute the value of $L_{max}(t+1)$ as it follows:

- the events t_i in case (1.a) do not change their contribution in $L_j^{max}(t)$. In fact, their contribution was already present in the computation of $L_{max}(t)$ because $t_i \in P_{max}$ and it continues to be present because $t_i \in B_{t+1}$.
- the events t_i in case (1.b) are introduced in the computation of $L_j^{max}(t)$.

Unfortunately, for those events that belong to E_t and E_{t+1} but not to $P_{max}(E_t)$ we can not claim anything. In fact, let us consider the precedence graph in Fig. 5.4(b). At instant $t = 3$ all are in E_3 but not in P_{max} because the overall contribution is negative. The situation does not change until $t < 5$. In fact, for $t = 5$ you have that $B_5 = \{t_1\}$ and $E_5 = \{t_2, t_3\}$, thus the value of ru_{1j} affects L_j^{max} and moreover both t_2 and t_3 are inserted into P_{max}; the overall contribution is now, $ru_{2j} + ru_{3j} = -2 + 7 = 5 > 0$, positive.

Even tough we cannot say anything about $t_i \notin P_{max}(E_t)$ and $t_i \in E_{t+1}$, it is possible to prove the following necessary condition:

Theorem 5.2 *An element* $t \notin P_{max}(E_t)$ *belongs to* $P_{max}(E_{t+1})$ ***only if*** *one of the following two conditions hold:*

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

Proof: We prove the two cases separately:
Case 1: if $t_i \in A_t \cap (E_{t+1} \cup B_{t+1})$ then a further element is added to P_{max} only if $ru_{ij} > 0$. Indeed if $ru_{ij} \leqslant 0$ then there exists at least one production t_k^+ that is implied by t_i^-. Thus it is possible to put only t_k^+ in the set P_{max} having a bigger value of L_{max}. Then $ru_{ij} > 0$.
Case 2: if it exists $t_i \in (E_t \backslash P_{max}(E_t)) \cap B_{t+1}$ then a further element is added to P_{max} only if $ru_{ij} < 0$. Indeed if $ru_{ij} > 0$ then there exists a time point t_k s.t. its contribution $\Sigma ru_{kj} > 0$ and the combined contribution of t_i and t_k is negative. But this is possible only if $\Sigma ru_{ij<0}$ that is at least a time point $t_z \in (E_t \backslash P_{max}(E_t)) \cap B_{t+1}$ s.t. $ru_{zj} < 0$. □

From the theorems introduced so far it is possible to deduce a set of rules which allow an improvement in the computational cost of computing the resource envelope reducing the number of times that it is necessary to recompute the set P_{max} (Theorem 5.1), and the size of set from which to extract it, from E_{t+1} to $E_{t+1} \backslash P_{max}(E_t)$ (Theorem 5.2). It is worth noting that this is also a fundamental improvement for solving a given problem instance with a variant of a PCP-like solver that incorporates resource envelopes for guidance.

Algorithm. Algorithm 5.1 describes how to incrementally compute the resource envelope for a resource r_j. In this case we only show how to compute the value of L_j^{max}. The algorithm can be easily modified to compute the value for L_j^{min}.

As explained above an important point is that L_j^{max} has been computed only at at time instants in which either B_t or E_t changes. For any time point t_i this can only happen either at its earliest time value $est(t_i) = -d(t_i, t_0)$ or at its latest time value $lst(t_i) = d(t_0, t_i)$. We collect this set of significant instants as the first step of the algorithm. In the main loop two aspects are taken in consideration:

- the events that move from E_{t-1} to B_t;
- the possible change of $E_t - P_{max}$.

These two aspects allow to reduce both the number of times a Max-Flow algorithm is requested and the size of the Max-Flow problem: from P_{max} to $E_{t-1} - P_{max}$.

For the first case the variables L_{max} and P_{max} are brought up to date easily. In the other case if the situation described in theorem 5.2 holds then a Max-Flow algorithm is requested to find possible new time points to insert into P_{max} and to update the value of L_j^{max}.

Evaluation. Summarizing in this section we have introduced a new incremental approach to compute the envelope for a stepwise-constant resource allocation. Unlike the method proposed in [Muscettola, 2002] we do not apply a complete algorithm to each time point but we try to take some advantages from the computation of the maximum resource level value in the previous time point.

The first important contribution is that we describe for all possible kinds of changes that can happen what to do and, moreover, just in some of these cases we need to apply a complex approach. Even though in the worst case the bounds value have to be computed at each time point, as in [Muscettola, 2002], the reader can note that the proposed method can be applied on a reduced number of events, namely those belonging to the set $E_t - P_{max}(E_{t-1})$ which is a subset of the set E_t taken into account in [Muscettola, 2002]. This is the second contribution of the present work.

In conclusion, the analysis presented in this section has given the following contributions:

Algorithm 5.1 Compute the Resource Envelope Incrementally

Input: A scheduling problem

Output: The resource envelope

collect and sort all the significant time instant, $est(t_i)$ and $lft(t_i)$ $\forall t_i$

compute B_0, E_0, and A_0

$L_j^{max}(-1) = \sum_{t_i \in B_0} ru_{ij}$

$P_{max} = \emptyset$

for all the significant time instant t **do**

$L_j^{max}(t) = L_j^{max}(t-1)$

update B_t, E_t and A_t

for each events t_i such that $(t_i \in B_t) \wedge (t_i \in E_{t-1})$ **do**

if $t_i \in P_{max}$ **then**

$P_{max} = P_{max} - \{t_1\}$

else

$L_j^{max}(t) = L_j^{max}(t) + ru_{ij}$

if $(\exists t_i : ru_{ij} > 0 \wedge t_i \in E_t \cap B_{t-1}) \vee (\exists t_i : ru_{ij} < 0 \wedge t_i \in B_t \cap (E_{t-1} - P_{max}))$

then

$\Delta P_{max} \leftarrow maxflow(E_{t-1} - P_{max})$

$P_{max} = P_{max} \cup \Delta P_{max}$

$L_j^{max}(t) = L_j^{max}(t) + \sum_{t_i \in \Delta P_{max}} ru_{ij}$

return L_j^{max}

- the use of the "maxflow" algorithm is not neccessary in all the time points. Indeed it is possible to apply a trivial operation to update the value of L_j^{max}
- the "maxflow" algorithm is applied on the problem associated to the set $E_t - P_{max}(E_{t-1})$ rather than the one associated to its superset E_t.

To evaluate the actual improvement obtainable using these two rules we have have carried out a set of experiments using the benchmark problems described in [Neumann and Schwindt, 1999]. This benchmark consists of fours sets of different size 10×5, 20×5, 50×5 and 100×5 (number of activities $\times$ number of resources). Figure 5.5 shows a comparison of the average CPU times required to compute the resource envelope for any instance of the benchmark. The three curves show the CPU time with the original approach, using the first rule and using both the incremental rules. It is evident how these rules are able to overcome the drawback of applying a Max Flow method at each step making it useful also for larger problems. In the Sect. 5.4.3 we also show the effects of these rules embedded in a resource envelope scheduler.

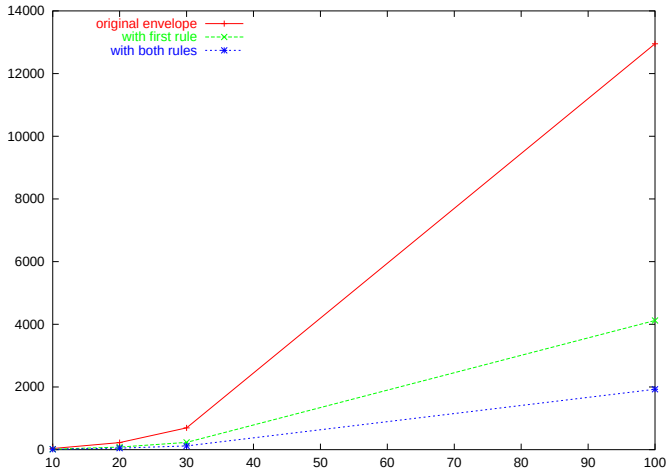


Figure 5.5: Comparison of the CPU times required to compute the resource envelope with the different methods

5.3.1 Other incremental approaches in the literature

Some other works have recently addressed the problem of reducing the high computational cost that the use of the resource envelope to guide the search process implies. In general, there are two places where incremental computation can be leveraged: (1) in the computation of the envelopes across points of discontinuity (for the same set of events and constraints), and (2) in the computation of the envelopes when new events and constraints (reflecting the refinement of partial plans) are added.

While our work follows the first approach, the work described in [Satish Kumar, 2003] investigates the second point. Starting from the maximum flow at one stage, the method consists in computing the maximum flow for the next stage by either minimally reducing the flow (if nodes and/or edges have been deleted) or minimally increasing the flow (in case new nodes and/or edges are posted). Even though the approach is based on the incremental computation of a Max-Flow problem, it does not consider the ordering of the time points present in the problem, and this leads to missing the improvement of the asymptotic complexity.

The work by [Muscettola, 2004] is on the same line as ours. In particular the approach described in the previous section and Muscettola's work has been developed independently and first presented at the same conference [Policella *et al.*, 2004b]. Muscettola describes similar properties and additionally proves that the combination of these properties with specific Max-Flow algorithms leads to a time complexity $O(Maxflow(n, m, U))$, where n is the number of nodes, m the number of constraints, U the maximum capacity of an edge from the source σ or to the sink τ, and

$Maxflow(n, m, U)$ is the time complexity of the Max-Flow algorithm.

It is worth noting as among the algorithm for which this results can be proved there is the preflow-push algorithm [Goldberg and Tarjan, 1988] that has been used in the actual implementation of our approach (for more details about the Max-Flow algorithms which respect the previous property see Table 1 and Theorem 12 in [Muscettola, 2004]).

5.4 EBA: the resource Envelope Based Algorithm

This section describes the first version of the Resource Envelope based scheduler, EBA. The algorithm has been implemented following the template described in Fig. 4.2. This consists in a greedy solver which iteratively analyzes the resource envelope and posts a new constraint. Three variants of this algorithm have been obtained using the different heuristics to solve contention peaks described in Sect. 4.3. In spite of its simplicity, this approach has been pursued to identify the drawbacks of using resource envelope based solver and to single out possible ways to solve them.

Even though different points, heuristics, resource envelope computation (incremental or not), have been already described, one aspect is still missing: how to detect a peak on a resource envelope. In Sect. 5.4.1 we discuss this aspect and analyze two different alternatives. We then give the results obtained applying the three different variants of the EBA algorithm to well known RCPSP/max benchmark sets.

5.4.1 Detecting peaks on resource envelopes

As mentioned before the resource envelope represents the tightest bound for the resource usage given a set of temporal alternatives[3] for each activity. In practice the envelope can be used as a means to analyze the current situation looking for possible flaws. Indeed, if the value of the resource envelope does not respect the capacity constraints then the current situation is not admissible, or more precisely, there exists at least a temporal solution $s \in S_T$ which is not feasible ($s \notin S$). At this point it is necessary to extract from the current situation the *contention peak* – i.e. a set of activities – which leads to a flaw.

A basic method to collect peaks is used in [Policella *et al.*, 2003] in the case that the activities require only renewable resources[4]. In this case the effect of an activity stops as soon as the its execution stops and then it is possible to find the activities which contribute to the peak by just looking at the time interval associated with an over-allocation (under-allocation). The method consists of the following three steps:

1. compute the resource envelope profile;

[3] This situation can be represented as a graph.

[4] This is the case of the RCPSP/max.

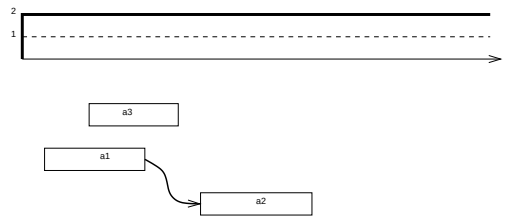


Figure 5.6: Resource Envelope: aliasing effect in the conflict collection.

2. detect intervals of over-allocation (under-allocation);

3. collect the set of activities which can be potentially executed in this interval.

Unfortunately this approach presents a major drawback. Indeed it can pick activities which are already ordered. This is a not negligible aspect, in fact once a contention peak is selected we look for any pair of its activities to find a new ordering precedence to post. Thus this way of selecting the peaks can lead to either consider already ordered activities or, worse, to post redundant constraints. For example, let us consider the following problem (see Fig. 5.6): a binary resource and a set of three activities a_1, a_2 and a_3. Any activity has the same interval of allocation, $[0, \infty)$, and moreover a precedence constraint between the pair (a_1, a_2) is defined, $a_1 \prec a_2$. In this case the previous method collects the peak $\{a_1, a_2, a_3\}$. This set implies to erroneously consider the pair (a_1, a_2) among the possible conflicting activities. Whereas the only two conflicts are (a_1, a_3) and (a_2, a_3).

To avoid this aliasing effect a more careful method has been introduced. This method derives from considering the sets B_t, E_t, A_t and $P_{max}(E_t)$ described in the previous section. This method is based on the particular assumption that each activity simply uses resources; without production and/or consumption. It is worth noting that the sets above are collected during the resource envelope computation thus their use does not imply any computational overhead.

Also for this enhanced method the two first steps coincide with the first two of the previous method: compute the resource envelope and match it with the resource capacity to find possible intervals of over-allocation. Then in the following step, the contention peaks are collected according to the following rules:

- pick any activity a_i such that the time point associated with its start time is in $P_{max}(E_t)$ whereas the time point associated with its end time is not in $P_{max}(E_t)$, that is:

$$contention\ peak = \{a_i | st_i \in P_{max}(E_t) \wedge et_i \notin P_{max}(E_t)\}.$$

- to avoid collecting redundant contention peaks, the collection of a new contention peak is performed only if:

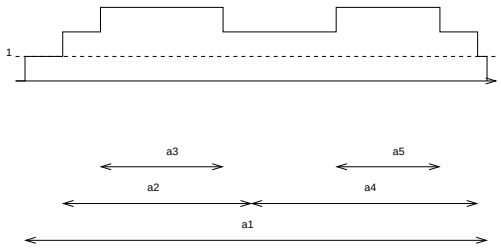


Figure 5.7: Resource Envelope: detection of maximal peaks.

(a) there exists at least an activity a_i such that the time point associated to its end time, et_i, moves from A_{t-1} to $B_t \cup E_t$ (i.e. $et_i \in A_{t-1} \cap (B_t \cup E_t)$);

(b) there exists at least an activity a_j such that the time point associated to its start time, st_j, moved in P_{max} since the last time a conflict peak has been collected.

The first rule is necessary to identify if an activity a_i is actually contributing to the value of L_j^{max}. In fact while the end time belongs to P_{max} the start time belongs to $P_{max} \cup B_t$. Thus the effects of the two events are balanced out, giving a void contribution to the resource envelope value. For instance in the case in Fig. 5.6 the more informed method selects the two peaks: $\{a_1, a_3\}$ and $\{a_2, a_3\}$.

The second rule is necessary to avoid the collection of redundant peaks. Let us consider the example in Fig. 5.7: here you are five activities each one requiring the same binary resource. The arrows represent the possible interval of allocation of each activity. If the peaks were collected considering only the interval of over allocation, we would have the following result: $\{a_1, a_2\}$, $\{a_1, a_2, a_3\}$, $\{a_1, a_4\}$ and $\{a_1, a_4, a_5\}$. It is possible to note that the first and the third set are subsets of, respectively, the second and the fourth. Considering instead the changes into both A_t and B_t, we are able to compute non-redundant sets, in the case of the example $\{a_1, a_2, a_3\}$ and $\{a_1, a_4, a_5\}$.

Algorithm 5.2 summarizes the approach to collect the conflicting peaks. It starts by sorting all the time points, then, it runs over them. If t_i is the start time of an activity and is also a new element of P_{max} then the activity is added to the set $Peak$. If t_i is instead the end time of an activity, if at least one new element has been added in $Peak$ since the last $Peak$ was collected, then $Peak$ is added into $Collected\_Peaks$. We conclude by recalling that in the results shown in the next section we consider only the second method to collect peaks.

Algorithm 5.2 Collecting Peak on the Resource Envelope

Input: the resource envelope for the current situation

Output: the set of contention peaks

sort the time points t_i
$Collected\_Peaks = \emptyset$
$Peak = \emptyset$
for all the time points t_i **do**
  if t_i is start time of a_k **then**
    opp
    if $t_i \in P_{max}(E_t) - P_{max}(E_{t-1})$ **then**
      $Peak = Peak \cup \{a_k\}$
      $new\_element = true$
  else
    if $t_i \in (A_{t-1} \cap (B_t \cup E_t))$ **then**
      if $new\_element$ **then**
        $Collected\_Peaks = Collected\_Peaks \cup \{Peak\}$
        $new\_element = false$
        $Peak = Peak - \{a_k\}$
return $Collected\_Peaks$

5.4.2 Results

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

In Sect. 3.5 we have introduced two metrics for robustness: $fldt$ and $flex$. Both these metrics are related with the number of feasible solutions *contained* in a $\mathcal{POS}$. In particular, $flex$ is directly related to the number of independent pairs of activities (no precedence constraint) in a partial order schedule. On the contrary, the disruptibility $dsrp$ is related with the *stability* of a solution, such that we consider executions where only one unexpected event at a time can occur (e.g., activity duration lasts longer than expected or the start time of an activity is shifted forward). We report as

[5]All the algorithms presented here are implemented in C++ and the CPU times presented in the following tables are obtained on a Pentium 4-1500 MHz processor under Windows XP.

	$\|\mathbf{flex}\|$			$\|\mathbf{fldt}\|$		
	J10	J20	J30	J10	J20	J30
EBA	0.14	0.16	0.23	0.63	0.64	0.69
EBA+MCS linear	0.17	0.15	0.18	0.65	0.60	0.59
EBA+MCS quadratic	0.16	0.13	0.16	0.65	0.58	0.56

Table 5.1: $|flex|$ and $|fldt|$

	$\|\mathbf{dsrp}\|$		
	J10	J20	J30
EBA	0.53	0.58	0.58
EBA+MCS linear	0.56	0.56	0.57
EBA+MCS quadratic	0.55	0.57	0.56

Table 5.2: $|dsrp|$

a result a value related to the average number of activities affected (number of start time changes) by the set of unexpected events.

Table 5.1 and 5.2 present the main results according to the three normalized values $|flex|$, $|fldt|$, and $|dsrp|$. In each case, the higher the value, the better the quality of the corresponding solutions. In addition, Table 5.3 complements our experimental analysis with four more results: (1) percentage of problems solved for each benchmark set, (2) average CPU-time in seconds spent to solve instances of the problem, (3) average minimum makespan and (4) the number of leveling constraints posted to solve a problem. It is important to observe that the three variants of the EBA algorithm present different degrees of effectiveness (see $\%solved$ in Table 5.3). Thus, to have a fair comparison among the different variants, the experimental results in this section are computed with respect to the subset of problem instances solved by all three approaches.

It is possible to note from the results shown in Table 5.1 that the simplest EBA variant overcomes the other two: in fact we found that in the case of the larger benchmark set, $J30$, its $|flex|$ value is 0.23 whereas it is 0.16 in the "quadratic MCS" variant. The gap becomes larger if we consider $|fldt|$: 0.69 versus 0.56. Unfortunately this result has to be reconsidered in the light of the different effectiveness of the EBA variants. Indeed even though the "quadratic MCS" variant presents worse quality values, it is able to solve almost twice the instances solved through the simplest variants.

The results obtained with respect to the disruptibility metric (Table 5.2) show

	%solved			**makespan**		
	J10	J20	J30	J10	J20	J30
EBA	77.04	50.74	43.33	58.31	96.48	118.17
EBA+MCS linear	85.19	71.11	68.89	55.29	92.65	112.14
EBA+MCS quadratic	97.78	89.63	82.22	55.47	94.03	116.10
	CPU-time (secs)			**posted constraints**		
	J10	J20	J30	J10	J20	J30
EBA	0.11	1.37	7.53	11.54	33.40	63.29
EBA+MCS linear	0.18	1.83	8.82	11.12	32.87	56.84
EBA+MCS quadratic	0.19	1.99	10.94	12.38	34.98	59.64

Table 5.3: Comparison of both the EBA approaches.

the same behavior of the three EBA variants, even if in this case the differences are less noticeable. On the other hand the use of more sophisticated heuristics requires a greater CPU time as it is possible to see in Table 5.1 but it also allows to obtain better makespan values, though the approach is not oriented to optimize it.

5.4.3 A note on envelope efficiency

We end the section with a final remark about our research goals. The main aim of this analysis has been producing robust solutions. In this respect, EBA is the first scheduling algorithm to integrate the recent research results on exact bound computation into a scheduling framework, and, in addition, we have improved the efficiency of the envelope computation considerably with respect to the very preliminary version presented in [Policella *et al.*, 2003].

We also want to underline the role in the reduction of computational cost played by the properties described in Sect. 5.3. Indeed, the computation of the envelope implies that it is necessary to solve a Max-Flow problem for each time-point. As indicated in [Muscettola, 2002], this leads to an overall complexity of $O(n^4)$ which can be reduced to $O(n^{2.5})$ in practical cases. These computational requirements at the present limit the effective application of the resource envelope[6]. The use of the incremental properties described in the previous section speeds up the solving process by avoiding re-computation of the envelope at each step of the search. Moreover Theorem 5.2 allows us to apply the Max-Flow algorithm to a subset of E_{t+1}: $E_{t+1} \backslash P_{max}(E_t)$.

[6] In the current implementation we use a Max-Flow method based on the *pre-flow* concept [Goldberg and Tarjan, 1988].

	$J10$		
	scratch	incremental	$\Delta\%$
EBA	0.21	0.11	48.1
EBA+MCS linear	0.40	0.18	55.8
EBA+MCS quadratic	0.41	0.19	53.2
	$J20$		
	scratch	incremental	$\Delta\%$
EBA	3.65	1.37	62.5
EBA+MCS linear	4.64	1.83	60.6
EBA+MCS quadratic	5.13	1.99	61.2
	$J30$		
	scratch	incremental	$\Delta\%$
EBA	15.37	7.53	51.0
EBA+MCS linear	23.27	8.82	62.1
EBA+MCS quadratic	30.47	10.94	64.1

Table 5.4: Comparison between the CPU-time (secs) required by the EBA approaches using both the incremental and no-incremental method for computing the resource envelope.

We have seen in Fig. 5.5 the effectiveness of the rules introduced in reducing the CPU time required to compute a resource envelope. Now, we show the benefits obtained using these rules in the whole solving process (Table 5.4). In particular, for each configuration of the EBA algorithm and each benchmark set ($J10$, $J20$ and $J30$) there are three different results: the average CPU-time in seconds for solving a benchmark set without incremental computation (*scratch* column), as the previous one but with the use of the incremental properties (*incremental* column) and the obtained percentage improvement over the non-incremental version of EBA ($\Delta\%$ column). The results confirm the effectiveness of the incremental computation which is able to improve the CPU-time from a minimum of 48.1% to a maximum of 64.1% over the scratch computation.

5.5 Increasing the efficiency of EBA

The results described in the previous sections have shown how the EBA methods are often not able to find a solution. In this section we introduce some remarks on the

method and we describe a possible expedient to overcome this inefficiency.

One of the problems of the previous approach lies in the heuristics used. Indeed they are often not able to select the "right" way toward a solution. Let us consider again how each heuristic works:

- first, the current situation is analyzed using the resource envelope;
- the contention peaks (temporal interval for which the the resource envelope is greater than the resource capacity) are collected;
- a new constraint is synthesized (and then posted) to remove one of the current peaks.

These three steps are repeated until a situation without any peak is reached. The main problem of this approach consists in the fact that the peaks selected at each step are hypothetical. In other words when a peak is detected it means that one of the possible temporal allocations of the activities creates a flaw: but not all.

In this hypothetical situation is difficult to find a search method able to pursue an effective set of decisions. In this respect, EBA methods are somewhat similar to partial order planning approaches for temporal domains. Also in the latter case a set of situations is taken into account during the solving process. Even though this approach allows to directly obtain flexible solutions, it turns out to be very ineffective, especially, if it is compared with "positioned constrained" planners [Do and Kambhampati, 2003]. In practice, if on one hand the knowledge can improve the efficiency of the solving process, on the other hand, too much knowledge can overwhelm the decision process and result in degraded performance (see for instance the results in [Zweben *et al.*, 1994]).

Thus one of the reasons for the lack of efficiency in the EBAmethods is the need to consider several possible situations at each step. To reduce the search space and then focus the attention of the heuristics toward more promising directions we enrich the EBA methods with a set of constraint propagation rules. These aim at reducing the search space by pruning some of the non-feasible alternatives present in the set of temporal solutions taken into account.

To improve the EBA approaches we also modify the procedures to explore a larger subset of the search space. The approach consists of an iterative application of the EBA method. In order to allow this iterative procedure to explore different portions of the search space we use randomized versions of the three heuristics.

In the remaining of the section we first recall the concept of constraint propagation and then the set of rules implemented. Afterward, we describe the template of the iterative procedure and, finally, the results of the new approach are discussed.

5.5.1 Constraint Propagation

In the definition of a problem, constraints are introduced to describe the set of consistent solutions. These constraints can be also used in an active mode to remove values from the domains of the variables, deduce new constraints and detect inconsistent decisions. This process is called Constraint Propagation.

As the RCPSP-max problem is NP-hard, constraint propagation cannot be complete. This means that some but not all the consequences of the set of constraints are deduced. In particular, constraint propagation cannot detect all the inconsistencies.

In the approaches described so far we have used one type of propagation: temporal propagation. In fact given a set of activities and a set of constraints among them, we extract all the consistent temporal solutions. This is possible because the temporal problem of a scheduling problem[7] is solvable in polynomial time. Indeed, it can be represented like a graph where each node is an activity and each temporal constraint is represented through an edge between the pair of nodes associated with the activities which it constrains[8]. Based on this representation, applying an "All pair shortest path" algorithm, it is possible to deduce the feasible domain for each activity.

A further reduction of the search space can be obtained by considering also the combinatorial aspects of a scheduling problem: the resource constraints. There are different techniques for resource propagation. The *timetabling* techniques consider the possible usage of the resource at any time t: the domain of any activity is restricted by removing the time instants that would lead to a violation of the resource constraints. A different kind of technique is based on the analysis of the possible iterations among activities: *edge-finding* and *energetic reasoning*. These techniques instead of analyzing the time instant t, consider a subset of activities competing for the same resource and perform propagation based on the relative position of each activity in the set. For a thorough analysis of constraint propagation and, more in general, on constraint programming approaches for scheduling problems we refer the reader to [Baptiste *et al.*, 2001]. A more specific analysis of existing approaches to propagate resource constraints can be found also in [Laborie, 2003]. In the remaining of the section we describe one of the propagation techniques introduced in [Laborie, 2003] which has been used in the EBA methods.

Balance Constraints

In this section we recall the balance constraints techniques as they are previously introduced in [Laborie, 2003].

The basic idea of the balance constraint is to compute, for any time point t_i, a lower and upper bound on the resource level just before and after t_i. This approach is based on a different partition with respect to the one in Fig. 5.2. In this case instead of

[7] i.e., a scheduling problem where there are only temporal constraints and no resource constraints.

[8] This representation is well known in the literature with the name "Activity on the node".

considering the time interval of any activity, the relations among them are considered. From this the following partition arises:

- $S(t_i)$, the set of time points simultaneous with t_i;
- $B(t_i)$, the set of time points which proceed t_i;
- $BS(t_i)$, the set of time points that can be scheduled before or in parallel with t_i;
- $A(t_i)$, the set of time points after t_i;
- $AS(t_i)$, the set of time points that can be scheduled after or in parallel with t_i;
- $U(t_i)$, the set of time points which cannot be ranked with respect to t_i.

Based on these sets it is possible to define four different bounds for the resource usage: $L^{<}_{max}(t_i)$, $L^{>}_{max}(t_i)$, $L^{<}_{min}(t_i)$ and $L^{>}_{min}(t_i)$, which represent respectively, the upper bound before and after t_i and the lower bound before and after t_i. For instance you have:

$$L^{<}_{max}(t_i) = \sum_{t_j \in P \cap (B(t_i) \cup BS(t_i) \cup U(t_i))} q(t_j) + \sum_{t_j \in C \cap B(t_i)} q(t_j)$$

where P and C are respectively the set of time points which have associated a production and a consumption of a resource[9].

Considering these bounds it is possible to discover new precedence relations among the activities. The previous formula can be re-written as it follows:

$$L^{<}_{max}(t_i) = \sum_{t_j \in B(t_i)} q(t_j) + \sum_{t_j \in P \cap (BS(t_i) \cup U(t_i))} q(t_j)$$

if the first term, $\sum_{t_j \in B(t_i)} q(t_j)$, is negative it means that some of the productions in $BS(t_i) \cup U(t_i)$ have to be scheduled before t_i to have a correct usage of the resource. Moreover, if there exists a time point $t_j \in BS(t_i) \cup U(t_i)$ such that

$$\sum_{t_z \in P \cap (BS(t_i) \cup U(t_i)) \cap (B(t_j) \cup BS(t_j) \cup U(t_j))} q(t_z) < - \sum_{t_j \in B(t_i)} q(t_j)$$

then, if we had the constraints $t_i \leq t_j$ we again would not have enough resource for the time point t_i. Thus the constraint $t_j < t_i$ is deduced.

[9] that is, $t_j \in P$ ($t_j \in C$) if $q(t_j) > 0$ ($q(t_j) < 0$).

Algorithm 5.3 *i*EBA template

Input: a problem P

Output: a partial order schedule

```
POS* ← ∅
loop
  POS ← EBA_rndm(P)
  if POS better than POS* then
    POS* ← POS
    if termination criterion = true then
      break
return POS*
```

5.5.2 An Iterative Sampling Procedure

In this section we describe an extension of the EBA methods introduced above, toward a multi-pass, iterative sampling procedure. This approach aims at improving the efficiency of the solving process increasing the number of solved problems. Furthermore, the possibility to choose among different solutions for the same problem allows to increase the quality of the solutions found.

Figure 5.3 describes the template for the iterative sampling procedure named *i*EBA. The algorithm consists in the iteration of the function $\text{EBA}_{rndm}(P)$ that is a randomized version – it uses a randomized version of the heuristics described above – of the EBA algorithm. Given a problem P, *i*EBA returns a partial order schedule, $\mathcal{POS}$, if it is able to find it, otherwise it returns the empty set. Even though this algorithm is not complete, it turns out to be very useful to face difficult problems like the ones which are the object of our study.

The randomized version of the EBA method is obtained by performing a pseudo-random selection of the decision [Oddi and Smith, 1997]. Given an heuristic h, any conflict c has an heuristic value $h(c)$. In the EBA methods the conflict which has the maximum heuristic value $h_{max}(c)$ is selected. In the case of the *i*EBA method we rather select, randomly, one among the best conflicts, with respect to their heuristic value. More precisely, given a value $0 \leq \beta \leq 1$ we randomly select one of the conflicts c_i among those which hold the following inequality:

$$(1 - \beta)h_{max}(c) \leq h(c_i) \leq h_{max}(c).$$

It is possible to note that if $\beta = 0$ the choice is completely random, whereas if $\beta = 1$ the best heuristic value is selected (as in the case of EBA). Different termination criteria can be used to stop the loop: for instance, the CPU time or the number of iterations. In our implementation we used the latter criterion. Moreover, in the empirical evaluation presented in the next section, the makespan value is used to select the best

solution.

The approach presented assures us to exploit a larger subset of the search space. Besides the necessity to explore different paths in the search space, the idea of using a pseudo random selection of the heuristics is useful for the following two factors: (1) different decisions can have the same heuristic value and in this case they are discriminated only for the way in which these decisions are ranked; (2) the selection of the best decision may not lead to the best solution. Indeed the random selection allows to overcome these two possible faults.

5.5.3 Results

In this section we discuss the results obtained using the iterative sampling version of the EBA method, *i*EBA. The results we show refer to the benchmark set $J30$. In producing the results we consider two different levels of iterations: 3 and 10 restarts. Moreover we compare also the results obtained using or not the "balance constraint" propagation rules described above. This, together with the use of three alternative heuristics, produces twelve different variants of the *i*EBA methods.

Table 5.5 shows the solving ability achieved using an iterative version of the EBA methods, *i*EBA, enriched with the use of the resource propagation techniques described above. We can compare these results with the ones obtained by using the three EBA variants (see Table 5.3).

%solved	3 restarts		10 restarts	
propagation	n	y	n	y
*i*EBA	44.81	47.78	46.67	48.89
*i*EBA+MCS linear	84.81	88.54	89.63	94.81
*i*EBA+MCS quadratic	92.59	94.44	95.56	97.41

Table 5.5: *i*EBAs efficiency with respect to the number of iterations and the use of resource propagation

It is possible to note that, as expected, using different restarts we are able to increase the number of solved problems with respect to the EBA variants. Moreover using the propagation rules we find a further improvement that ranges from 1% to about 6%, depending to the method and on the number of iterations.

Considering the single variant of the *i*EBA method, we can notice (first row in Table 5.5) that in case we use the simplest heuristic (i.e. the one without MCS) the percentage of solved problem is still less than 50%: also the use of the propagation rules does not give a noticeable improvement (slightly greater than 1%).

In the case of *i*EBA+MCS linear we have the great improvement both with respect to the result obtained by EBA+MCS linear, and with respect to the use of resource

propagation. Indeed we found an improvement which ranges from 16%, in the case of the 3 restart no propagation variant, to 26%, in case of the 10 restarts with resource propagation variant. Furthermore, it is possible to note that in this case the use of the resource propagation gives the greater difference over all the results. Indeed we have a mean difference between using or not the resource propagation of about 5%.

As far as the last variant is concerned, we have that even though it gives the best value in terms of solved problems it does not benefit from the iterative procedure and the propagation rules, as the previous variant.

CPU time (secs)	3 restarts		10 restarts	
propagation	n	y	n	y
*i*EBA	17.049	15.945	56.83	53.15
*i*EBA+MCS linear	16.629	16.563	55.43	55.21
*i*EBA+MCS quadratic	17.799	17.586	59.33	58.62

Table 5.6: CPU time

Table 5.6 shows the CPU time values for the different variants of the *i*EBA method. It is important to note that the use of constraint propagation increases the time requested. This aspect can be neglected in the case of *i*EBA+MCS linear where the use of resource propagation also increases the efficiency from 88.51 to 94.81 in the case of 10 restarts. This is less acceptable in the case of variants which use the quadratic MCSIn fact in this case the efficiency is already high and propagation rules give a low improvement.

$\|\mathbf{flex}\|$	3 restarts		10 restarts	
propagation	n	y	n	y
*i*EBA	0.15	0.15	0.15	0.15
*i*EBA+MCS linear	0.12	0.12	0.12	0.12
*i*EBA+MCS quadratic	0.13	0.12	0.14	0.14

Table 5.7: Flexibility results

Table 5.7, 5.8 and 5.9 present respectively the results obtained for flexibility ($flex$), fluidity ($fldt$) and makespan (mk) (we remind the reader that for all three metrics, the higher the better). From these it is possible to notice different aspects:

1. for the $flex$ metric it is possible to note that the results obtained with the new methods are generally worse than the EBA variants. In fact, for instance, we have that the EBA+linear MCS has a value of 0.18, whereas the iterative version, *i*EBA+linear MCS , presents a result equals to 0.12 in the case of the 10 restarts variant that uses the propagation rules. The result stems from the use of the

makespan criterion to evaluate the different partial order schedules produced during the main loop of the algorithm in Fig. 5.3. Also comparing the different results obtained according to the number of restarts (3 or 10) and the use of resource propagation, we can see that there are no big variations.

$\|\mathbf{fldt}\|$	3 restarts		10 restarts	
propagation	n	y	n	y
*i*EBA	0.59	0.59	0.57	0.57
*i*EBA+MCS linear	0.61	0.60	0.59	0.59
*i*EBA+MCS quadratic	0.60	0.59	0.59	0.59

Table 5.8: Fluidity results

2. Considering the $fldt$ metric we have, instead, a preservation of the quality of the solutions found. We have for both the MCS variants of the iterative method *i*EBA better values than the ones obtained for EBA methods. In case of *i*EBA with quadratic MCS (3 restarts no propagation version) we have an improvement from 0.56 to 0.60. A different aspect can be noticed observing the use of propagation rules. In fact, these turn out to produce partial order schedules with a lower fluidity factor.

mk	3 restarts		10 restarts	
propagation	n	y	n	y
*i*EBA	106.41	106.38	106.92	106.38
*i*EBA+MCS linear	103.75	103.38	103.05	102.2
*i*EBA+MCS quadratic	102.44	102.13	101.76	101.35

Table 5.9: Makespan values

3. Table 5.9 presents the average makespan values obtained. We have that the makespan is improved with respect the non-iterative version. In case of *i*EBA + quadratic MCS with 3 restarts and propagation there is an improvement of about 13% with respect to the non-iterative version (EBA+MCS quadratic in Table 5.3). Regarding the different makespan results obtained according to the use, or not, of propagation, it is possible to see that, even though the use of resource propagation helps to achieve better makespan values, there is only a slight improvement.

5.6 Conclusions

In this chapter we have described a least commitment approach for generating flexible solutions. To obtain a least commitment approach we have used the resource envelope recently introduced by Muscettola. This allows to compute the minimal ad maximal bounds for the resource usage considering all the possible temporal solutions admissible by a time ordering of the activities. This has produced the first scheduler which integrates the resource envelope knowledge in the solving process. The use of the resource envelope knowledge has required/allowed the study of different related aspects:

- How to boost the envelope computation: in Sect 5.3 a set of properties to reduce the complexity of computing the resource envelope have been described.
- How to manage the resource envelope to extract real conflicts: in Sect 5.4.1 we have underlined how a superficial analysis of the resource envelope can produce aliasing effects in identifying the activities that produce resource conflicts.
- Pruning useless information through constraint propagation: the use of resource constraint propagation has been investigated to discover possible unfeasible solutions and increase the efficiency of using the resource envelope.
- More intensive analysis of the search space. To improve the solving process a more intensive algorithm has been implemented to overcome the possible lack of potency of heuristics choices.

To better understand the prospects of this approach it is worth comparing it with a different method. In the next chapter we see the results that can be obtained using a different approach for producing flexible solutions.

Chapter 6

Solve & Robustify

This chapter presents a different approach for building partial order schedules. Basically, this new method consists in building a flexible solution starting from a fixed-time schedule, where a start time value for every activity is defined. We show here that starting from a single solution of the scheduling problem it is possible to generate a set of solutions in the form of a Partial Order Schedule, $\mathcal{POS}$.

This alternative way of building flexible solutions has been motivated by the need to exploit the characteristics of the fixed-time solutions to produce flexible solutions. In fact preserving the quality of the fixed-time solutions in the final partial order schedule, can give an appealing result. We show that the procedure used to obtain a flexible solution maintains the original fixed-time solution among the set of schedules described by the $\mathcal{POS}$. This guarantees that in the best case possible – i.e., no disruptions happen – the characteristics of the fixed-time solution are, at least, preserved.

6.1 Introduction

Different needs – and, dually, different trade-offs between quality and computational times – are addressed by different algorithms or combinations of solving techniques in a meta-heuristic schema. The Solve & Robustify approach consists in a two step procedure where fixed-time solutions built in a first phase are used to obtain partial order schedules in an ensuing step. The proposed approach analyzes the two aspects of finding a solution and building a flexible schedule, separately.

Basically, the approach is based on the assumption of independence between the classical scheduling objectives – like minimizing the makespan – and the need to obtain a robust (or flexible) solution. Even though this assumption is in general not true, the two step approach can allow to exploit *state-of-the-art* schedulers to ob-

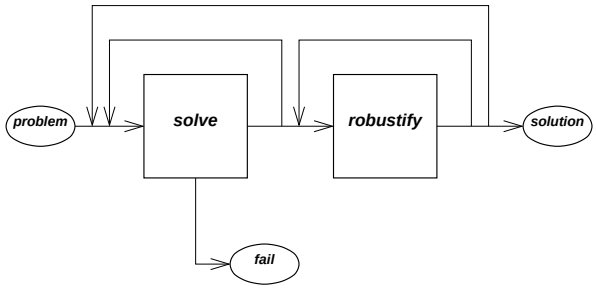


Figure 6.1: The general two step – solve & robustify – schemata

tain optimal solutions with respect to the classical objectives. Therefore in a next step a flexible solution is generated trying to preserve the optimality of the starting schedule. Preserving the optimality can be very important when a low degree of uncertainty is present. In this case the actual execution of the problem remains "close" to the *expected value problem* (i.e., the problem as described in input), therefore, the characteristics of a $\mathcal{POS}$ tend to maintain, due to necessary repairs, the new allocation of the activities close to the original schedule. It is intuitive that the closer the two allocations the fewer the loss in objective function optimality.

Figure 6.1 shows a possible sketch of the two step approach. This is based on the use of two separate modules: a greedy solver that has the aim of finding a first fixed-time solution, and a "robustify" module where a partial order schedule is synthesized. Different variants of the two step approach can be obtained by different combinations of the two modules:

- find a solution through the greedy solver and then robustify it;
- find a solution through the greedy solver and then explore the space of all the $\mathcal{POS}$s obtainable from the starting solution and select the one which optimizes a particular robustness metric;
- iterate a pseudo-random version of the greedy module to optimize the fixed-time solution with respect to a specific metric. In the next step, robustify this optimized solution;
- iterate a pseudo-random version of the greedy module to optimize the fixed-time solution with respect to a specific metric. In the next step, explore the space of all the $\mathcal{POS}$s obtainable from this optimized solution and select the one which optimizes a particular robustness metric;
- for each fixed-time solution generated by the pseudo-random version of the greedy algorithm find a $\mathcal{POS}$ and select the one which optimizes a particular robustness metric;

- for each fixed-time solution generated by the pseudo-random version of the greedy algorithm explore the space of all $\mathcal{POS}$s achievable from the fixed-time solution and select the one which optimizes a particular robustness metric;

As the figure highlights, only in the first phase, in which the search for a solution occurs, it is possible to fail (i.e., when it is not possible to find a first fixed-time schedule), whereas in the Robustify step, whenever a starting schedule exists, it is always possible to generate a partial order schedule.

A further remark concerns the generation of flexible schedules. In the case of multi-capacitive resource problems, we have in general more than one possible $\mathcal{POS}$ corresponding to a fixed-time schedule. On the contrary in the case of binary resources, i.e. job shop problem, a fixed-time solution gives also a unique "linearization" of all the activities. This aspect has supported and suggested the idea of exploring the space of possible $\mathcal{POS}$s obtainable from the same fixed-time schedule with the aim of increasing robustness characteristics.

In the remaining of the chapter we describe the implementation of the introduced approach and the results obtained. Section 6.2 introduces a first, greedy, schemata for the two step approach. Then in the following sections more intensive methods are described and analyzed. But first, we conclude this introduction giving some further comments about how the the Solve-and-Robustify compares with the envelope based approaches described in Chapter 5.

Solve & Robustify vs. Envelope based Approaches

Figure 6.2 gives a sketched view of both the approaches with respect to the search space. The different sets represent temporal solutions (they are subsets of S_T) associated to a graph obtainable by posting new constraint on the initial scheduling problem.

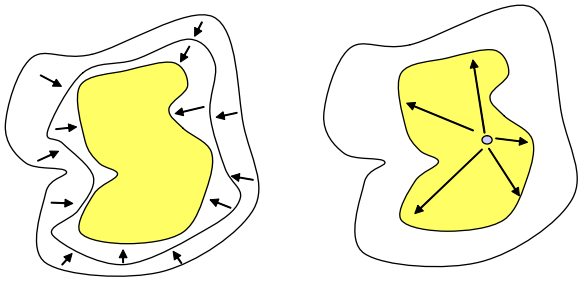


Figure 6.2: Solve & Robustify vs. Envelope based Approaches

The use of the resource envelope (on the left hand picture) implies an iterative reduction of the space of all the possible temporal solutions until this set contains only

feasible solutions. On the contrary, the two-step procedure first computes a single solution (a point in the search space) then generalizes the result to obtain a set of solutions (see the right hand picture in Fig. 6.2). Thus on one hand the envelope-based approach considers all the temporal solutions at each stage and tries to select decisions which reduce as less as possible this set (i.e. least commitment). Conversely, in the two step procedure the final objective, i.e. to obtain a partial order schedule, is neglected in the first step where the actual goal is to find a feasible fixed-time solution. Only in the successive phase the aim of building flexible solutions is taken under consideration.

6.2 Coupling a solver with a robustify step

This section describes the implementation of the first alternative of the list introduced in Sect. 6.1. The approach is based on the use of a greedy algorithm to find a first – fixed-time – solution. Therefore a post processing procedure (that is named *chaining*) is applied to obtain a partial order schedule. It is worth noting that it is possible to use different methods for both producing the initial fixed-time solution and for generating the flexible solution.

The section is organized in the following way: first, the greedy solver used to produce fixed-time solutions is described. Then the following subsection introduces the *chaining* procedure, the method used to synthesize flexible solutions starting from a fixed-time schedule. The section ends showing the results obtained using this method.

6.2.1 The Earliest Start Time Algorithm - ESTA

The Earliest Start Time Algorithm – ESTA – is a greedy solver previously introduced in [Cesta *et al.*, 1998]. This algorithm has been used in the actual implementation of a first solve and robustify method [Policella *et al.*, 2004b]. The algorithm consists in a Precedence Constraint Posting approach (see Fig. 4.2). Like the envelope based methods it consists in the following three steps: evaluate the current situation; identify possible flaws (or conflicts) and the reasons which cause them; select and solve one conflict by posting a new constraint. These steps are repeated until a solution – situation without any flaw – is found. The main difference between Envelope based methods and ESTA is in the use of a different resource profile to evaluate the current situation. In fact, the ESTA algorithm reasons on the *earliest start time profile*. This profile is obtained considering any activity allocated in its earliest start time.

Definition 6.1 (Earliest Start Time Profile) *Let $est(t_i)$ be the earliest time value for the time point t_i. For each resource r_j we define the Earliest Start Time Profile as*

the function:

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

We recall that given a temporal constraint problem, that is a set of activities and a set of temporal constraints[1], the earliest and the latest start time for any activity can be obtained through a polynomial algorithm based on the computation of the shortest and longest path between a (dummy) source – posted at the origin – and any activity. This can be accomplished with a computational complexity $O(n^2)$ by using Dijkstra's algorithm [Dijkstra, 1959]. Furthermore it is possible to prove that two solutions of the temporal problem can be obtained allocating all the activities to either their earliest or their latest start time[2].

In the ESTA method the resource profile is computed according to one precise fixed-time solution: the Earliest Start Time Solution. Therefore, unlike the Resource Envelope, it analyzes a well-defined scenario rather than the range of all possible temporal behaviors. It is worth noting that the key difference between the two approaches is that while the latter gives a measure in the worst-case hypothesis, the former identifies "actual" conflicts in a particular situation. In other words, the first approach says what *may* happen in such a situation relative to the entire set of possible solutions, the second one, instead, what *will* happen in such a particular case. Even though, using a precedence constraint posting method to solve the problem, we always have a set of temporally consistent solutions S_T, the ESTA method ensures resource-consistency of only one of these solutions (i.e., $S_T \nsubseteq \mathcal{S}$). Below, we describe a method for overcoming this limitation and generalizing an early start time solution into a partial ordered schedule.

Likewise the resource envelope analysis, in the particular case of activities that do not consume/produce resources, we use a peak detection approach to avoid sampling redundant peaks. Specifically, we have followed the strategy described in [Cesta *et al.*, 2002] that collects sets of maximal peaks, i.e., sets of activities such that none of the sets is a subset of the others. Even though this approach allows to avoid the collection of redundant peaks, it is still possible to have a same MCS in two (or more) different peaks. Furthemore, like the EBA method, also for ESTA we have three versions according to which of the three heuristics introduced in Sect. 4.3 is used to guide the search process.

6.2.2 Producing a Partial Order Schedule with Chaining

A first method for producing flexible solutions from an earliest start time solution has been introduced in [Cesta *et al.*, 1998]. This consists in producing a flexible solution where a *chain* of activities is associated with each unit of each resource. In this

[1] See inequalities (3.1) and (3.2).

[2] Further details about temporal constraint problem can be found in [Dechter *et al.*, 1991].

Algorithm 6.1 Chaining procedure

Input: a problem P and one of its fixed-time schedules S

Output: A partial order solution $\mathcal{POS}$

$\mathcal{POS} \leftarrow P$
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**
      $k \leftarrow SelectChain(a_i, r_j)$
      $a_k \leftarrow last(k)$
      $AddConstraint(POS^{ch}, a_k \prec a_i)$
      $last(k) \leftarrow a_i$
return $\mathcal{POS}$

section we generalize that method for the more complex resource model included in the scheduling problem considered here (RCPSP/max). Given a solution, a transformation method, named *chaining*, is defined that proceeds to create sets of chains of activities. Algorithm 6.1 describes the chaining process. This operation is accomplished by deleting all previously posted leveling constraints and using the resource profiles of the earliest start solution to post a new set of constraints.

The concept of chaining form refers to a $\mathcal{POS}$ 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 $a_{j,i-1}$ is 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. In this 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 is allocated to as many chains as necessary to fulfill its resource requirements.

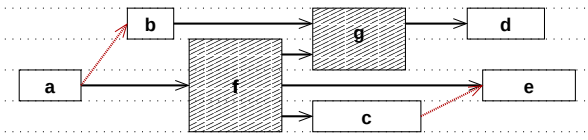


Figure 6.3: A partial order schedule in chaining form

The example in Fig. 6.3 represents a partial order schedule for a problem with a single resource r_k with capacity $c_k = 4$. The bold edges represent the set of chains and the thin edges 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 require more than one unit of resource. This implies that both of them are allocated to more than one chain. Algorithm 6.1 uses a set of queues, $queue_{jk}$, to represent each capacity unit of the resource r_j. The algorithm starts by sorting the set of activities according to their start time in the solution S. Then it proceeds to allocate the capacity units needed for each activity. It selects only the capacity units available at the start time of the activity. Then when an activity is allocated to a queue, a new constraint between this activity and the previous one in the queue is posted. Let m and max_{cap} be respectively the number of resources and the maximum capacity among the resources, the complexity of the chaining algorithm is $O(n \log n + n \cdot m \cdot max_{cap})$ where n is the number of activities.

By coupling the chaining method with the ESTA algorithm described above it is possible to obtain three different algorithm variants according to the heuristic used in the ESTA algorithm (see Sect. 4.3):

- ESTA^C – the ESTA algorithm uses the slack based heuristic as it has been described in [Smith and Cheng, 1993];

- ESTA^C+MCS linear – the ESTA algorithm uses the MCS linear sampling;

- ESTA^C+MCS quadratic – the ESTA algorithm uses the MCS quadratic sampling.

In the next section we compare the results obtained by using these algorithms with the three variants of the EBA algorithm.

Before concluding we want to remark that concepts similar to the idea of a *Chaining Form* schedule have also been used elsewhere: for example, the Transportation Network introduced in [Artigues and Roubellat, 2000], and the Resource Flow Network described in [Leus and Herroelen, 2004] 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 *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 enough temporal slack allowing chained activities to move “back and forth”.

6.2.3 The ESTA^C algorithm: Results

In this section we compare the results obtained in Sect. 5.4.2 with the three ESTA^C variants. This comparison is obtained using the same benchmark problems $J10$, $J20$ and $J30$. Also in this case in order to produce an evaluation of the three metrics $fldt$, $flex$ and $dsrp$ that is independent from the problem dimension, we show their normalized value (see formula (3.5)).

	$\|\mathbf{flex}\|$			$\|\mathbf{fldt}\|$		
	J10	J20	J30	J10	J20	J30
EBA	0.14	0.16	0.23	0.63	0.64	0.69
EBA+MCS linear	0.17	0.15	0.18	0.65	0.60	0.59
EBA+MCS quadratic	0.16	0.13	0.16	0.65	0.58	0.56
ESTA^C	0.19	0.20	0.25	0.67	0.65	0.60
ESTA^C+MCS linear	0.20	0.20	0.26	0.66	0.65	0.62
ESTA^C+MCS quadratic	0.20	0.19	0.25	0.68	0.64	0.64

Table 6.1: $|flex|$, $|fldt|$.

	$\|\mathbf{dsrp}\|$		
	J10	J20	J30
EBA	0.53	0.58	0.58
EBA+MCS linear	0.56	0.56	0.57
EBA+MCS quadratic	0.55	0.57	0.56
ESTA^C	0.64	0.74	0.73
ESTA^C+MCS linear	0.65	0.73	0.72
ESTA^C+MCS quadratic	0.65	0.72	0.72

Table 6.2: $|dsrp|$.

First, we observe that all six tested strategies are not able to solve all the problems in the benchmark sets $J10$, $J20$ and $J30$. The first column of Table 6.3 shows the percentage of solved problems by each strategy. This observation is particularly important, because the rest of the experimental results in this section are computed with respect to the subset of problem instances solved by all six approaches (both EBA's and ESTA^C's variants).

Table 6.1 and Table 6.2 present the main results for the six different approaches, according to the three incremental metrics introduced above. In each case, the higher

	% solved			**makespan**		
	J10	J20	J30	J10	J20	J30
EBA	77.04	50.74	43.33	58.31	96.48	118.17
EBA+MCS linear	85.19	71.11	68.89	55.29	92.65	112.14
EBA+MCS quadratic	97.78	89.63	82.22	55.47	94.03	116.10
ESTA^C	96.30	95.56	96.30	47.35	72.90	79.21
ESTA^C+MCS linear	98.15	96.67	96.67	46.63	72.45	78.45
ESTA^C+MCS quadratic	98.15	96.67	97.04	46.70	72.75	78.55
	CPU-time (secs)			**posted constraints**		
	J10	J20	J30	J10	J20	J30
EBA	0.11	1.37	7.53	11.54	33.40	63.29
EBA+MCS linear	0.18	1.83	8.82	11.12	32.87	56.84
EBA+MCS quadratic	0.19	1.99	10.94	12.38	34.98	59.64
ESTA^C	0.02	0.12	0.41	6.40	18.69	35.10
ESTA^C+MCS linear	0.03	0.18	0.74	6.23	17.49	34.07
ESTA^C+MCS quadratic	0.03	0.19	0.83	6.26	17.40	34.00

Table 6.3: Comparison of both the EBA and the ESTA approaches.

the values, the better the quality of the corresponding solutions. In addition, Table 6.3 complements our experimental analysis with four more results: (1) percentage of problems solved for each benchmark set, (2) average CPU-time in seconds spent to solve instances of the problem, (3) average minimum makespan and (4) the number of leveling constraints posted to solve a problem.

From Table 6.1 we first observe that the ESTA^C approaches dominate the basic EBA approaches across all problem sets for the two metrics directly correlated to solution robustness. This observation is confirmed in Table 6.1 where better values of flexibility correspond to better values of disruptibility (stability). Hence, the solutions created with ESTA^C are more appropriate to absorb unexpected events.

This fact induces further observations about the basic strategies behind the two algorithms. EBA removes all possible resource conflicts from a problem $\mathcal{P}$ by posting precedence constraints and relying on an envelope computation that produces the *tightest* possible resource-level bounds for a flexible schedule. When these bounds are less than or equal to the resource capacities, we have a resource-feasible partial order ready to *face* with uncertainty. However, in order to remove all possible conflicts EBA has to impose more precedence constraints than ESTA^C (see column labeled with *posted constraints* in Table 6.3), with the risk of overcommitment in the final solution. In fact, in comparing EBA with ESTA^C, it can be seen that the EBA approach

is actually less effective. It solves significantly fewer problems than ESTA^C in each problem set, obtains solutions with higher makespans, incurs into higher CPU times and posts more precedence constraints. By adding MCS analysis to the EBA search configuration, we obtain a noticeable improvement of the results. In fact, in the case of quadratic sampling the number of problems solved is closer to that achieved with the ESTA^C approach. However, we pay a higher price in terms of CPU time and there are no significant improvements in the makespan and in the number of constraints posted (Table 6.3).

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

6.3 Partial order schedules in chaining form

In the previous section we showed that the two step approach is capable of generating partial order schedules more efficiently than a least commitment $\mathcal{POS}$ generation procedure while simultaneously producing solutions 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 generating $\mathcal{POS}$s.

Our goal in this section is to examine the concept of *Chaining Form* solutions from the broader perspective of generating robust $\mathcal{POS}$s and to establish properties that can guide the development of chaining procedures capable of generating more robust partial order schedules. We describe a canonical graph form, the *Chaining Form* ($\mathcal{POS}^{ch}$) for representing a partial order schedule and show that any given $\mathcal{POS}$ 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 such a form. This is then accomplished introducing a family of operators for transforming a generic fixed-time schedule into a partial order schedule in chaining form, $\mathcal{POS}^{ch}$. Before proceeding, we recall that in our approach a solution is robust if it allows a rapid system reconfiguration whenever an unforeseen event happens.

We start by considering a flexible solution obtained by the chaining procedure: indeed, by definition, a solution in chaining form is a partial order schedule. It is also possible to prove that any partial order schedule $\mathcal{POS}$ admits at least an equivalent

$\mathcal{POS}$ in chaining form[3].

Theorem 6.1 *Given a partial order schedule $\mathcal{POS}$ there exists a partial order schedule in chaining form, $\mathcal{POS}^{ch}$, that represents at least the same set of solutions.*

Proof. Let $\overline{POS}(V_P, \overline{E})$ be the transitive closure of the graph $\mathcal{POS}$ 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 $\mathcal{POS}^{ch}(V_P, E_P \cup E^{ch})$ with $E^{ch} \subseteq \overline{E}$ such that $\mathcal{POS}^{ch}$ 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 $E_k^{ch} \subseteq \overline{E}$ 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 (possibly empty) set 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 $\mathcal{POS}^{ch}$. 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 $\mathcal{POS}$ is a partial order schedule. Thus, it is always possible to build a $\mathcal{POS}^{ch}$ from a $\mathcal{POS}$ with $E^{ch} \subseteq \overline{E}$. □

Given this result, we can restrict our attention, without loss of generality, to the set of $\mathcal{POS}$s which are in chaining form. Hence, a general operator for transforming a fixed-time schedule into a $\mathcal{POS}$ can be defined as follows:

Definition 6.2 (Chaining operator) *Given a fixed-time schedule S a chaining operator $ch()$ is an operator that applied to S returns a partial order schedule*

$$\mathcal{POS}_S^{ch} = ch(S)$$

such that $\mathcal{POS}_S^{ch}$ is in chaining form and S is contained in the set of solutions it describes.

Algorithm 6.1 describes a basic chaining operator. The function $SelectChain(a_i, r_j)$ represents the core of this procedure; it admits different definitions giving different results. In basic implementation it chooses, for each activity, the first available chain

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

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 is always 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 a resource feasible (fixed-time) solution has been built. Given that a common objective of the first step in many scheduling domains is to construct a feasible fixed-time solution that minimizes makespan, the following property plays an important role:

Property 6.1 *Given a fixed-time schedule S and its $\mathcal{POS}_s^{ch}$*

$$mk(ES(\mathcal{POS}_s^{ch})) \leq mk(S) \tag{6.1}$$

that is, the makespan of the earliest solution of $\mathcal{POS}_S^{ch}$ is not greater than the makespan of the input solution S.

Equation (6.1) can be explained considering that by definition S is one of the solutions represented by $\mathcal{POS}_S^{ch}$. Practically, since only simple precedence constraints already contained in the input solution S are added, the makespan of the output solution is not 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.

Because the original solution is always included in the generated $\mathcal{POS}$ it is intuitive that in the produced partial order schedule there exists at least one fixed-time solution which preserves the original characteristics. Moreover, given an objective function μ, it can be proved that the Earliest Start Time solution of a given $\mathcal{POS}$ is such that:

$$\mu(ES(\mathcal{POS}_s^{ch})) \leq \mu(S) \tag{6.2}$$

This is because the objective functions take into account how far an activity is allocated in time and any activity in the Earliest Start Time solution of a $\mathcal{POS}$ is not allocated later than in the original fixed-time solution.

6.3.1 Remarks on the chaining method

A common way to approach scheduling problems is based on considering a set of identical binary resources[4] like a single multi-capacitive resource. This section aims at describing the role that this aspect can have on the synthesis of scheduling problem solutions and, in particular, of robust schedules.

To illustrate more clearly the resource abstraction aspect we consider the following example composed of a set of tasks, or activities, which require one machine to

[4]That is, with $max_k = 1$.

be processed, among a set of l identical machines. This problem can be modeled as a set of activities which require one unit of a multi-capacitive resource. In other words, instead of considering a set of binary resources $R' = \{r_{1'}, \ldots, r_{l'}\}$, a set with a unique resource $R'' = \{r_{1''}\}$, with $max_{1''} = l$, is taken into account.

This resource abstraction turns out to be very useful to approach scheduling problems. Indeed it avoids taking into account complicated disjunctive constraints, simplifying the solving process. Let us consider the previous example: in case a task a_i requires a single machine without the resource abstraction it is necessary to consider the following constraint:

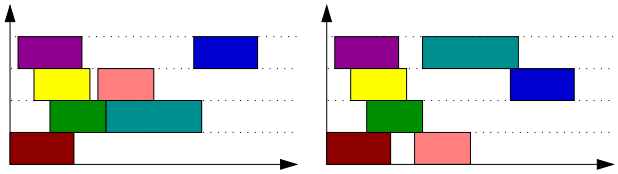
$$\begin{gathered}(r_{i1'} = 1 \wedge r_{i2'} = 0 \wedge \ldots \wedge r_{il'} = 0) \vee \\ (r_{i1'} = 0 \wedge r_{i2'} = 1 \wedge \ldots \wedge r_{il'} = 0) \vee \\ \vdots \\ (r_{i1'} = 0 \wedge r_{i2'} = 0 \wedge \ldots \wedge r_{il'} = 1)\end{gathered}$$

which represents the fact that one and only one resource (machine) is necessary for processing activity a_i. On the contrary, to substitute the set of binary resources with a cumulative resource it is sufficient to only specify the requirement $r_{i1''} = 1$.

This resource abstraction on the binary resources allows to postpone the allocation of each scheduled activity on a sub-set of the resources. In fact, to execute a schedule it is necessary to know which resources are assigned to each activity. The schedule resulting from the first step, defines a temporal allocation for any activity and ensures that for each of them there exists a set of available machines that can be accommodated, but specifies nothing about which machine has to be used. For this reason it is necessary to introduce this step of "resource allocation", in which the binary resources are consistently allocated to the whole set of activities. The two steps, scheduling and resource allocation, work on two different levels of abstraction. It is worth noting that while a failure is possible during the first phase, it is always possible to generate a resource allocation starting from a schedule. In fact, a resource allocation for a schedule can be easily computed with a polynomial algorithm assigning each activity to the first available resource.

Despite its simplicity, the resource allocation step can play a fundamental role in generating robust schedules. Let us consider the following example: Fig. 6.4 shows two different resource allocations for a common schedule. On the y axis the different binary resources (machines in the case of the example above) are represented. From the graph it is possible to extract the usage timetable of each resource. As noted earlier, it is clear that this representation is robust if and only if there is some temporal slack that allows chained activities to move "back and forth".

Analyzing the two solutions it is possible to note that in the solution of Fig. 6.4(a) there is a critical allocation on the second binary resource (second row from below), as the starting time of the second activity coincides with the completion time of the



(a) (b)

Figure 6.4: Two different resource allocations of a common schedule.

first one. Thus any possible delay of the first activity, during the execution phase, implies an equivalent delay of the second activity. On the contrary the resource allocation in Fig. 6.4(b) ensures a minimal slack between any pair of consecutive activities. The presence of these slacks allows to absorb possible disruption. Obviously the ability to absorb a delay is directly proportional to the extent of the slack.

On the basis of this example it should be clear how an "intelligent" resource allocation method may lead to an increase of the robustness of a schedule. In the example for instance the method should lead to the second allocation rather than the first one. As it is shown in Fig. 6.4 the second step (resource allocation or robustify) may take place by properly distributing the available binary resources over all the activities, leaving their starting times untouched.

6.4 Increasing Robustness Features through Iterative Chaining

In Sect. 6.1 we introduced the idea of an iterative method to robustify a given fixed-time schedule (see Fig. 6.1). Based on this goal the previous section has introduced two important aspects:

- given a fixed-time solution, it is possible to generate one or more $\mathcal{POS}$s or, in other words, it is possible to obtain different resource allocations;
- given a resource allocation (or a $\mathcal{POS}$) generated from a fixed-time schedule, it is alway possible to generate it with a chaining method or, in other words, there exists an equivalent chaining form for the $\mathcal{POS}$.

These two aspects support the introduction of an iterative chaining method. The chaining operator introduced in the previous section transforms a feasible fixed-time solution into a $\mathcal{POS}$ in chaining form by dispatching activities to specific resource units[5]. In the basic implementation (Algorithm 6.1) this dispatching process is carried out in a specific deterministic manner; the $SelectChain(a_i, r_j)$ sub-procedure

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

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 concerning how to dispatch activities to resource units, it is possible to generate different $\mathcal{POS}$s from a given initial fixed-time 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 of the final $\mathcal{POS}$ produced by the chaining process. 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 in chaining form. Specifically, the chaining operator (Algorithm 6.1) is executed n 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 generated $\mathcal{POS}$ 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 certainly examines a large number of candidate $\mathcal{POS}$s, it does so in an undirected way and this is likely to limit overall search effectiveness. More effective procedures are obtained by using heuristics to bias the way in which chains are built. Of course the use of heuristics requires an analysis of the chaining form solution with the aim of finding properties which can give an increase in the effectiveness of the solution produced. These properties are used to implement new chaining operators. The next subsection has these aims, while, in Sect. 6.4.2, we show the results obtained using these operators.

6.4.1 Generating different Partial Order Schedules

To design a more informed heuristic for dispatching activities to chains, it is necessary to examine the structure of solutions produced by the chaining procedure. Let us start by considering the example in Fig. 6.3. We note that both the activities require 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 mutually constrains two, otherwise independent processes. When an unforeseen event occurs and must be taken into account, the presence of these points works 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. A synchronization point originates from one of two different sources:

- a constraint defined in the problem which relates pairs of activities belongs to different chains;
- an activity that requires two or more resource units and/or two or more resources has to be allocated on two or more chains.

In the first case, the synchronization point is a strict consequence of the scheduling 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. 6.5. Here a more flexible solution than the one previously discussed in Fig. 6.3 is obtained by simply allocating the two gray activities to the same subset of chains. In the $\mathcal{POS}$ in Fig. 6.3 the two gray activities span all four chains. They effectively split the solution into two parts, and the whole execution phase depends 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.

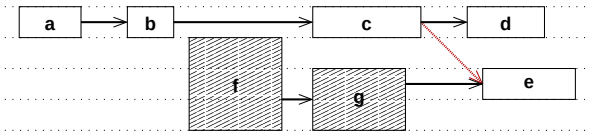


Figure 6.5: A more flexible $\mathcal{POS}$

Based on this observation, we define a first heuristic chain selection procedure that favors allocation of activities to common chains: MAXCC, *maximizing 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 the 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. 6.3. 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 forces the 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. 6.5.

The example in Figure 6.5 allows us to show 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 these problem constraints cannot be eliminated, they can in fact be made redundant if both activities can be allocated to the same chain(s).

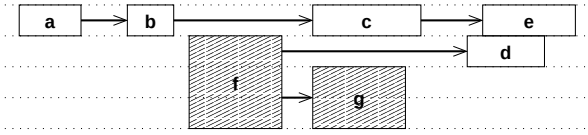


Figure 6.6: A even more flexible $\mathcal{POS}$

This observation leads to the definition of a second heuristic chain selection procedure: MINID, *minimizing interdependencies*. This 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 with respect to the 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

(**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 6.6 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. 6.5 is eliminated.

	$\|\mathbf{flex}\|$	$\|\mathbf{fldt}\|$	**cpu**	**npc**	**mk**
ESTA^{C}	0.25	0.59	0.41	38.7	107.1
ESTA^{iC}+$rndm_{flex}$	0.28	0.62	1.28	44.8	106.8
ESTA^{iC}+MAXCC$_{flex}$	0.33	0.63	1.27	39.1	106.7
ESTA^{iC}+MINID$_{flex}$	0.48	0.67	1.21	28.3	105.7
ESTA^{iC}+$rndm_{fldt}$	0.26	0.63	1.28	44.7	106.1
ESTA^{iC}+MAXCC$_{fldt}$	0.31	0.65	1.27	39.6	106.4
ESTA^{iC}+MINID$_{fldt}$	0.46	0.70	1.21	28.9	105.4

Table 6.4: Performance of the algorithms

6.4.2 Results

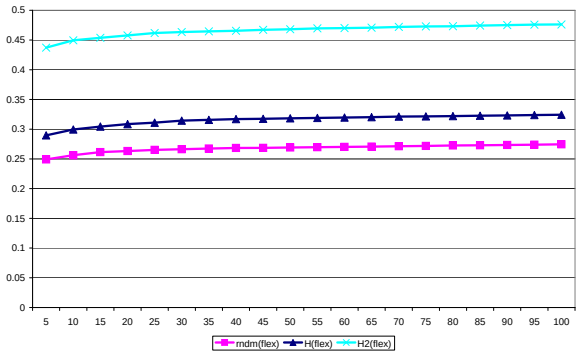
Table 6.4 summarizes the main results, in particular we compare the following three sets of chaining methods applied at the same solving process, ESTA:

- the basic chaining operator described in Algorithm 6.1, ESTA^{C};
- the iterative chaining procedure[6] which maximizes only the flexibility metric, $flex$. There are three different variants: the pure randomized version, ESTA^{iC} +$rndm_{flex}$, the first heuristic biased version aimed at maximizing chain overlap between activities that require multiple resource units, ESTA^{iC} +MAXCC$_{flex}$, and the enhanced heuristic biased version which adds consideration of current activity ordering constraints, ESTA^{iC} +MINID$_{flex}$.
- same as above with the difference that the optimized metric is the fluidity, $fldt$. In this case the procedures are named respectively, ESTA^{iC}+$rndm_{fldt}$, ESTA^{iC}+MAXCC$_{fldt}$ and ESTA^{iC}+MINID$_{fldt}$.

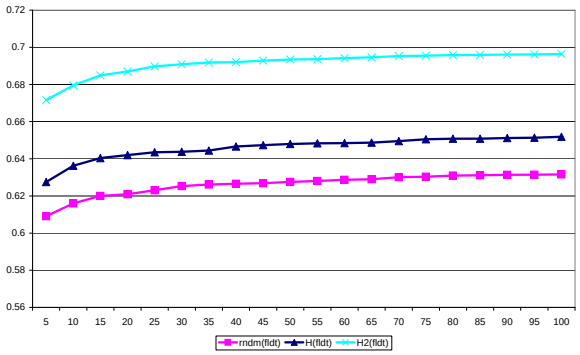
The results shown in Table 6.4 are the average values obtained over the subset of solved problems in the $J30$ benchmark. For each procedure the values of five metrics are shown: flexibility ($|flex|$), fluidity ($|fldt|$), CPU-time in seconds (cpu), the number of precedence constraints posted (npc) and the makespan (mk). With respect to CPU time, we include both the time to find an initial fixed-time 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 $\mathcal{POS}$s derivable from a given fixed-time solution S if the goal is to maximize solution robustness. In fact, all search strategies are also seen to produce some amount of

[6]For this reason we use the label iC instead of C.



(a) Flexibility vs number of iterations



(b) Fluidity vs number of iterations

Figure 6.7: ESTA^{iC}: efficiency of iterative sampling

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, ESTA^{iC}+MAXCC$_{flex}$, is able to improve 30% over the basic chaining results while the version using the enhanced heuristic, ESTA^{iC}+MINID$_{flex}$, obtains a gain of about 90% (from 0.25 to 0.48). 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 $fldt$ metric. In this case, improvement ranges from about 10% for the first heuristic ESTA^{iC}+MAXCC$_{fldt}$, to about 18%, for ESTA^{iC}+MINID$_{fldt}$. It is worth noting that in this case we achieve the best value of fluidity also with respect the previous results (0.70 versus the 0.69 obtained with the EBA algorithm).

Another discriminating aspect of the performance of ESTA^{iC}+MINID$_{fldt}$ is its ability of taking advantage of pre-existing precedence constraints and of reducing the number of posted constraints[7] (npc column in Table 6.4). This effect, as might have been predicted, improves 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 a compression of the critical path. On the other hand, the use of an iterative procedure incurs into a non negligible additional computational cost.

Figure 6.7 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. 6.7(a) represents the results obtained when the metric $flex$ is taken into account while in Fig. 6.7(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 6.4). For instance the best solution obtained by ESTA^{iC}+MAXCC$_{flex}$ after 10 iterations is better than the solution obtained by ESTA$^{iC}+rndm_{flex}$ after 100 iterations (see Fig. 6.7(a)); likewise, ESTA^{iC}+MINID$_{flex}$ (ESTA^{iC}+MINID$_{fldt}$) are able to obtain better solutions that can be obtained by any other procedure 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 the generation of better solutions.

6.5 Investigating the use of different fixed-time solutions

Section 6.1 has introduced the idea of coupling two modules to obtain partial order schedules. The ingredients of this approach are a solving method (to obtain fixed-time schedules) and a robustify phase to feed flexibility – to safeguard against execution uncertainty – into the original solution. This section aims at investigating the potentiality of the use of different fixed-time solutions for the same scheduling problem in order to achieve better robustness values. In the following we present two possible alternatives among those described by Fig. 6.1:

1. an algorithm composed by a first step in which an optimum makespan schedule is searched for; then one of the chaining methods described so far is applied on the best solution found;

2. an algorithm which iteratively builds a fixed-time solution and then produces

[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.

Algorithm 6.2 ISES^{iC}

Input: a problem P

Output: a partial order schedule $\mathcal{POS}$

```
S* ← ∅
loop
  find a new fixed-time schedule, S
  if S better than S* then
    S* ← S
  if termination criterion is true then
    break
POS ← chaining (S*)
return POS
```

a partial order schedule. Finally, the $\mathcal{POS}$ with the best robustness values is selected.

As the chaining method can be seen as a local search procedure, the last approach can be seen as a "Greedy Randomized Adaptive Search Procedure", or GRASP (for a survey on this meta-heuristic see [Resende and Ribeiro, 2002]). This approach allows to consider both the effectiveness of the solving phase and the efficiency of the robustify step. This is accomplished by integrating the two goals in the same algorithm. In the first approach, the ISES algorithm is used as the solving method [Cesta *et al.*, 1999]. This gives us two different algorithm ISES^{iC} and GRASP^{iC} , where the superscripted iC means that an iterative chaining method is used to obtain flexible solutions. Algorithms 6.2 and 6.3 show two template schemata for, respectively, the iterative and the grasp-like procedure.

6.5.1 The Iterative Sampling Procedure

In this section we recall the iterative procedure ISES as it has been previously described in [Cesta *et al.*, 1999]. This algorithm consists in an iterative, randomized, version of the ESTA algorithm described in Sect. 6.2.1. The iteration is used to overcome the lack of efficiency that a greedy algorithm can present. Even though it is not a complete algorithm the iterative procedure can give noticeable results for problems for which, being hard to solve (i.e. NP-hard problems), a complete algorithm is not practically useful [Motwani and Raghavan, 1995].

Algorithm 6.2 gives an high level view of the method. It iteratively computes a new schedule S, and compares it with the current best found solution. In case the new new solution is better then the current best one the last one is updated to the former. The loop continues while the termination criterion is not satisfied. In the

Algorithm 6.3 GRASP^{iC}

Input: a problem P

Output: a partial order schedule $\mathcal{POS}^*$

```
POS* ← ∅
loop
  find a new fixed-time schedule, S
  POS ← chaining (S)
  if POS better than POS* then
    POS* ←POS
  if termination criterion is true then
    break
return POS*
```

case of ISES the termination criterion consists of a predefined number of iterations while two solutions are compared with respect to their makespan.

The randomized version of ESTA is obtained by performing a pseudo-random selection of the decision. Given a heuristic h, any conflict c has a heuristic value $h(c)$. In the ESTA methods the conflict which has the maximum heuristic value $h_{max}(c)$ is selected. In the case of the ISES method one among the best conflicts with respect their heuristic value is randomly selected. More precisely, given a value $0 \leq \beta \leq 1$ we randomly select one of the conflicts c_i among those for which the following inequality holds:

$$(1 - \beta)h_{max}(c) \leq h(c_i) \leq h_{max}(c).$$

It is possible to note that in case $\beta = 0$ the choice is completely random, whereas if $\beta = 1$ the best heuristic value is selected (as in the case of EBA).

In the following the results of three different variants of the ISES algorithm are presented. These variants are obtained using the three different heuristics introduced in Sect. 4.

6.5.2 The Grasp-Chaining

The Greedy Randomized Adaptive Search Procedure, GRASP, is a multi-start meta-heuristic for combinatorial problems, in which each iteration consists basically of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result.

The two phases, construction and local search, clearly match with the two steps of our approach, solve and robustify. Therefore a GRASP procedure to generate partial order schedules can be obtained using a randomized version of our greedy algorithm,

ESTA, and a chaining operator. Like in the case of ISES, a randomized version of the greedy algorithm is necessary to explore different branches of the search tree.

Algorithm 6.3 describes a GRASP approach based on the Solve and Robustify schemata of Fig. 6.1. At each step, after a fixed-time schedule is found, a new partial order schedule is generated (by using either the simple or the iterative chaining operator). Based on a robustness metric the best found $\mathcal{POS}$ is returned.

It is worth noting the difference between the iterative procedure and the GRASP approach. While in the first a solution is generated from the best fixed-time solution found in the latter different $\mathcal{POS}$s are generated from different starting points (fixed-time solutions).

6.5.3 Results

In this section we describe the results obtained applying the two approaches, ISES^{iC} and GRASP^{iC}, to the J30 benchmark. For each method we have obtained six different variants according to both the heuristic chosen to guide the search[8] and to the criterion to optimize. In both cases the iterative chaining method with MINID heuristic is used to generate the final partial order schedule.

ISES^{iC}. Table 6.5 and Table 6.6 contain the results obtained using the six versions of the ISES^{iC} method. All the versions in a first phase find a first solution. In this phase the ISES algorithm is used considering as termination criterion 10 restarts. Next to obtain a partial order schedule the iterative chaining method is used with a number of iterations equals to either 10 or 100.

An analysis of these results shows us that the values of both the robustness metrics do not show great difference. In fact, in case of 10 restarts, the values of $|flex|$ is 0.45 while the values of $|fldt|$ range from 0.61 to 0.62. Of course, a major effort in terms of chaining restarts gives better results (0.48 in case of $|flex|$ and 0.63 in case of $|fldt|$).

If we compare the results obtained by using the the ISES^{iC} variants with the previous results shown in the chapter it is possible to notice a twofold behavior for the two robustness metrics. In fact in the case of $flex$, the values are much better than the ones obtained using both the simple chaining operator introduced in Sect. 6.2.2 and the randomized iterative sampling: the values in these cases are respectively, 0.25 and 0.28. Moreover the new values obtained are also very close to the best value 0.48 obtained in the case of using MINID with 100 iterations[9]. This result becomes more noticeable if the CPU time values (in seconds) requested in the two cases are taken under consideration: respectively 8.22 for the ISES^{iC}+MINID$_{flex}$ +quadratic MCS version and 1.21 for ESTA^{iC}+MINID. This is because in both the procedure the

[8] With or without MCS computation, linear or quadratic.

[9] We achieve this result in case of ESTA^{iC}+MINID$_{flex}$ +MCS quadratic with 100 iterations

	|flex|	**|fldt|**	**cpu**	**npc**	**mk**
ISES^{iC}+MINID$_{flex}$	0.45	0.60	4.97	30.47	98.70
ISES^{iC}+MINID$_{flex}$ +MCS linear	0.45	0.60	6.17	30.53	98.21
ISES^{iC}+MINID$_{flex}$ +MCS quadratic	0.45	0.60	7.08	30.50	98.23
ISES^{iC}+MINID$_{fldt}$	0.43	0.62	4.96	30.95	98.64
ISES^{iC}+MINID$_{fldt}$ +MCS linear	0.43	0.62	6.15	30.94	98.22
ISES^{iC}+MINID$_{fldt}$ +MCS quadratic	0.43	0.61	7.08	30.82	98.26

Table 6.5: ISES^{iC} 10+10

	|flex|	**|fldt|**	**cpu**	**npc**	**mk**
ISES^{iC}+MINID$_{flex}$	0.47	0.60	6.63	30.20	98.52
ISES^{iC}+MINID$_{flex}$ +MCS linear	0.47	0.60	8.22	30.26	98.12
ISES^{iC}+MINID$_{flex}$ +MCS quadratic	0.48	0.60	9.44	30.09	98.26
ISES^{iC}+MINID$_{fldt}$	0.45	0.63	6.63	30.64	98.44
ISES^{iC}+MINID$_{fldt}$ +MCS linear	0.45	0.63	8.22	30.64	98.10
ISES^{iC}+MINID$_{fldt}$ +MCS quadratic	0.45	0.63	9.42	30.74	98.22

Table 6.6: ISES^{iC} 10+100

iterative chaining method with MINID has an important rule in increasing the flexibility (in terms of $flex$) of the final partial order schedule.

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 fact the $fldt$ values showed in Table 6.5 are lower than the ones obtained with the randomize iterative procedures (with or without the heuristic guidance). The results obtained point to different aspects: first of all the use of more optimized initial schedules 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 solutions.

A less intuitive aspect is the different trade-off between the makespan value and both the robustness metrics. From the results shown previously it is possible to note the even though the values are worse than in the previous cases, the ISES^{iC} variants achieve $flex$ values close to the best value found. This is not the case in which the $fldt$ is the objective of the optimization. In practice while the ISES^{iC}+MINID procedure is efficient in increasing the $flex$ value, the same procedure is not able to find a "good" partial order schedule for the $fldt$ metric. This behavior is justified from the different nature of the two metrics: $flex$ is a qualitative criterion whereas

	|flex|	**|fldt|**	**cpu**	**npc**	**mk**
GRASP^{iC}+MINID$_{flex}$	0.46	0.67	6.43	29.17	105.25
GRASP^{iC}+MINID$_{flex}$ +MCS linear	0.47	0.67	7.49	28.54	104.65
GRASP^{iC}+MINID$_{flex}$ +MCS quadratic	0.47	0.67	8.54	28.85	104.86
GRASP^{iC}+MINID$_{fldt}$	0.44	0.69	6.43	29.36	105.08
GRASP^{iC}+MINID$_{fldt}$ +MCS linear	0.45	0.69	7.49	28.85	104.56
GRASP^{iC}+MINID$_{fldt}$ +MCS quadratic	0.45	0.69	8.53	28.99	104.59

Table 6.7: GRASP^{iC}

the $fldt$ is a more quantitative metric. Therefore a schedule with a better makespan value presents a more compact allocation of the activities allowing a lower degree of intervention to the iterative chaining procedure with the MINID heuristic. This aspect is further analyzed in Sect 6.6.

GRASP^{iC}. The GRASP^{iC} method represents the most intensive approach we have investigated. Its implementation is based on 10 iterations of the main loop, where an initial fixed-time schedule is computed at each step, and on 10 iterations on the inner loop, where a different $\mathcal{POS}$ is computed at each step. Thus the best found $\mathcal{POS}$ is selected among a set of 100 alternatives.

Table 6.7 shows the results obtained according to both the optimization criteria ($flex$ or $fldt$) and the chosen heuristic. Even though the values obtained for the two metrics are similar to the best obtained in Sect. 6.4, the approach it is not efficient if it is considered in the light of the CPU time requested. In fact these values are much greater than those requested by ESTA^{iC}+MINID: about 6.5 seconds versus 1.21 seconds respectively.

On the other hand, it is worth emphasizing that the use of more intensive approaches like ISES^{iC} and GRASP^{iC} are necessary not only for the optimization aspect but mostly to overcome the lack of efficiency that the use of a simple greedy procedure (like ESTA in Sect. 6.2.1) can entail.

6.6 Makespan versus robustness

In the previous section we have seen the results of the ISES^{iC} method which is based on the use of the ISES method to optimize the makespan value of the initial schedule as well as to increase the efficiency of the solving process. These results have shown that the optimization of the makespan value reduces the ability to obtain robust schedules especially if the fluidity metric is taken into account. In this section

we reconsider this result giving an insight into the nature of this effect.

Figure 6.8 underlines the motivations which lead to different behaviors between flexibility and fluidity with respect to the makespan optimization factor. In Fig. 6.8(a) the problem is represented. This is a one resource problem in which three activities have to be scheduled. These are ordered according to the constraints in the figure: between a and b there is a simple precedence constraint[10] 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 activity a. 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 equals to 3 and a resource requirement of 2. To complete the description of the problem, the resource capacity is equals to 2.

Figure 6.8(b) and 6.8(c) show two different solutions 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 both solutions there is a unique $\mathcal{POS}$: 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.

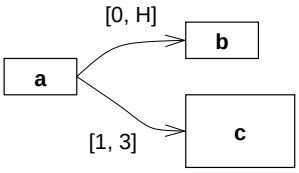
Let us consider first the flexibility metric. If we look at the two partial order schedules (on the right hand of Fig. 6.8(b) and 6.8(c)) it is possible to notice that in both cases there is the same $flex$ value. In fact in both cases the $flex$ value is zero because there is no pair of un-ordered activities.

Now we can shift our 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. 6.8(b) and $dist(a, b) = [4, H - 4]$ in the case in Fig. 6.8(c). This clearly shows that the flexibility value for the sub-optimal solution is greater than the one in Fig. 6.8(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. 6.8(b) while for the case in Fig. 6.8(c) $dist(c, b) = [0, H - 8]$

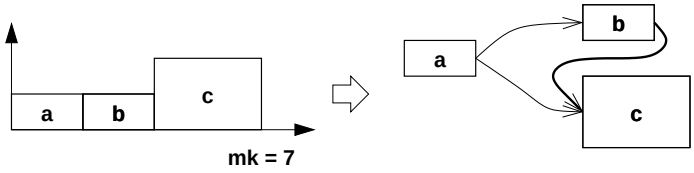
The problem is that in the schedule with the optimum makespan (Fig. 6.8(b)) the activity b is "caged in" by the other two activities. Therefore the time window constraint defined between a and c has the effect of limiting the flexibility of activity b. Furthermore since the capacity of the resource is equal to the requirement of c, no chaining method can overcome this problem.

It is possible to note that this is a peculiar characteristic of 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

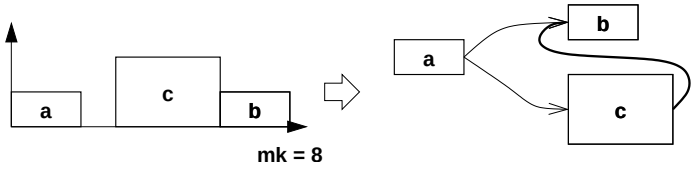
[10] Where H represents the temporal horizon of the problem.



(a) Initial problem



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



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

Figure 6.8: Example: dependency between makespan and robustness

solution.

6.7 Conclusions

In this chapter we have seen a different approach to generate flexible solutions. This is based on the coupling of two core modules: a *solver*, used to produce a fixed-time solution for the scheduling problem, and a *robustifier* which feeds flexibility into the schedule produced in the first phase (see Fig. 6.1). Moreover it has been shown that the procedure used to obtain the final solution always preserves the characteristics of the initial fixed-time schedule.

The simple pipeline exhibits a better behavior, with respect the robustness criteria, than the envelope based approaches (Sect. 6.2). Furthermore the possibility of using *state-of-the-art* schedulers allows to have a greater efficiency. In fact, as pointed out in Fig. 6.1 whenever a fixed-time solution is produced in the first phase, it is always possible to generate a partial order schedule.

An improvement on the robustness qualities has been obtained using more intensive search methods and heuristics which exploit the structure of the solutions.

Especially, the latter aspect has produced methods which present a good trade-off between quality and CPU time requested.

Finally, we have investigated the relation between the makespan and robustness criteria. We have found that this trade off can have different behaviors with respect to both the scheduling problem and the metric chosen as a yardstick for the robustness-related aspects.

Chapter 7

Discussion and Final Analysis

The aim of this chapter is to provide a final analysis and discussion of the partial order schedule paradigm as well as the approaches used to produce this kind of solutions. Specifically, we start with an analysis of another aspect of robust solutions: the ability of preserving a solution's stability. In fact, when it is possible to respond to external changes, maintaining the schedule close to the initial solution is important in order to avoid system "nervousness".

Next, in order to deepen the evaluation of the least commitment and solve-and-robustify approaches, we present the results obtained on a benchmark set of RCPSP/max instances which are larger (100 activities) than the ones used in the previous chapters (10 to 30 activities).

Finally, we conclude presenting a discussion about the limitations and the issues related to the research work shown in this dissertation (the $\mathcal{POS}$ paradigm and the solving methods).

7.1 Stability Analysis

In the previous chapters we have evaluated the different methods presented according to two metrics: flexibility ($flex$) and fluidity ($fldt$). These measures have been used to obtain an assessment of the qualities of the produced solutions ($\mathcal{POS}$s) in terms of the ability to absorb possible, unexpected, events. In fact, as already mentioned, in case of very complex scheduling problems (e.g. RCPSP/max) answers to unforeseen events without requiring a new scheduling phase, becomes of fundamental importance.

Nevertheless, when a solution is repaired it is possible to have several changes in it; this may produce an instability of the solution and nervousness in the execution. Furthermore preserving the solution's stability also has a different meaning; in fact a

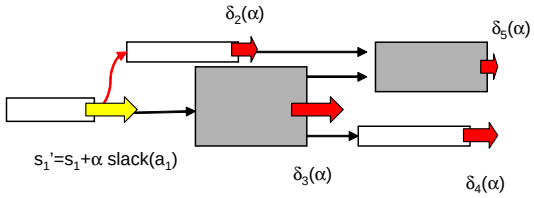


Figure 7.1: Stability evaluation: the impact of the shift $\alpha slack(a_i)$ over the activities is represented by the values $\delta_j(\alpha)$.

repair to an exogenous event is accomplished by adding a further set of constraints and transforming the actual $\mathcal{POS}$ into a new one. In this perspective it is easy to see that maintaining solution's stability equates to keeping as many solutions as possible in the final $\mathcal{POS}$ (which, in turn, implies having more alternatives for future changes).

For these reasons, a further analysis of the methods presented along the dissertation is necessary; this has been accomplished by introducing a further metric to measure stability, or, conversely, sensitivity, of the solutions produced. 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. 7.1): the start time s_i of a single activity a_i with *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 $0 \leq \alpha \leq 1$. This change may produce variations of the start time of the remaining activities a_j: $\delta_j(\alpha)$. We consider the mean value over the set of activities normalized with respect the window of possible start times of the activity a_i. A more operative definition of stability is given by the following formula:

$$stby(\alpha) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{\delta_j(\alpha)}{slack(a_i)} \tag{7.1}$$

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 increase of start time $\alpha slack(a_i)$ is performed on each activity's 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.

%α	**1**	**2**	**4**	**8**	**16**	**20**	**25**	**50**	**75**	**100**
ESTA^C	0.06	0.10	0.15	0.20	0.24	0.26	0.27	0.30	0.31	0.32
ESTA^C MCS l.	0.06	0.10	0.15	0.20	0.24	0.26	0.27	0.30	0.31	0.32
ESTA^C MCS q.	0.07	0.11	0.16	0.20	0.24	0.25	0.26	0.28	0.29	0.30
EBA	0.03	0.05	0.07	0.09	0.10	0.10	0.11	0.12	0.14	0.14
EBA MCS l.	0.04	0.06	0.10	0.14	0.18	0.18	0.19	0.22	0.24	0.25
EBA MCS q.	0.04	0.07	0.11	0.16	0.19	0.20	0.21	0.24	0.25	0.27

Table 7.1: Stability evaluation for ESTA^C and EBA variants on the benchmark set j30.

Results. Table 7.1 presents the results obtained evaluating the stability for the most simple algorithms described above (EBA and ESTA^C variants) on the benchmark set $j30$. 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 is obtained relaxing the resource constraints of the single instances and evaluating the stability of the resulting temporal net[1]. Conversely an upper bound for the stability values is represented by α, i.e. $stby(\alpha) \leq \alpha$. The values presented represent the average stability values of the solutions obtained through the various methods. Also in this case, to have a fair comparison we have considered the subset of common solved instances.

From the analysis of the data in Table 7.1, it is possible to note that the solutions obtained using the EBA variants present a better behavior with regard to the partial order schedules obtained through the two-step approaches. Indeed the solutions obtained by the EBA methods achieve the best quality values (i.e., 0.14 for $\alpha = 1$). About this result it is necessary to observe the following points: the makespan of the solutions obtained using EBA is larger than the values obtained for the ESTA^C variants; thus each activity in the former solutions has a larger time window of allocation, which allows a lower sensitivity to changes and thus a greater stability. The second point is that the EBA variant has a very low efficiency: in fact, it is able to solve just 43.33% instances of the benchmark $j30$ (see Table 6.3).

Table 7.2 presents the stability results on the subset of solutions obtained using respectively *i*EBA, ESTA^{iC} with the enhanced heuristic MINID, and GRASP^{iC}. This comparison is motivated on one hand by examining methods that have a similar efficiency (the *i*EBA is the more efficient method among the those based on the resource envelope analysis), whereas on the other hand we have considered two solve-and-robustify approaches to analyze the influence of an "intelligent" chaining process. The data show that when the efficiency of the envelope based analysis is improved,

[1]Once the resource constraints are removed the resulting graph constitutes a partial order schedule.

%α	1	2	4	8	16	20	25	50	75	100
*i*EBA true	0.06	0.09	0.13	0.18	0.21	0.22	0.24	0.28	0.30	0.32
ESTA^C + +MINID	0.04	0.06	0.10	0.14	0.17	0.18	0.19	0.22	0.24	0.25
GRASP^{iC}	0.04	0.07	0.10	0.14	0.18	0.19	0.20	0.22	0.24	0.25

Table 7.2: Stability evaluation for improved ESTA^C and EBA variants on the benchmark set j30.

we have a contemporary reduction of the quality of the solutions produced (0.32). On the contrary, the use of the enhanced heuristic to produce $\mathcal{POS}$s reduces the sensitivity of the solutions to single changes (from 0.30 to 0.25). This result is to be expected: in fact the heuristics aim at removing possible interdependencies among chains; this allows to have solutions in which each chain of activities can be executed asynchronously with respect to the remaining ones. This aspect reduces the sensitivity of the solution with regard to possible changes increasing the stability of the $\mathcal{POS}$.

The results in this section show that the solve-and-robustify method is able to produce stable $\mathcal{POS}$s. This adds to the efficiency of the solve-and-robustify paradigm, the ability of obtaining good makespan solutions in low CPU-time, and, last but not least, maintaining the high performance in terms of flexibility or fluidity shown in the previous chapters.

7.2 A large scale experimentation

A possible remark that can be done considering the empirical evaluations provided in the previous chapters, is that the use of a single set of benchmark problems can restrict the validity of the results. In particular the size of the problems (30 activities and 5 resources in the worst case) can suggest the investigation of larger problems. This section tackles this aspect presenting a further evaluation of the envelope based analysis and the solve-and-robustify paradigm on a larger benchmark $j100$ introduced in [Schwindt, 1998]; each of the 540 instances of this benchmark is composed of 100 activities that can require the use (one or more units) of 5 multi-capacitive resources.

In particular in this section we consider the greedy versions of the two methodologies (the ESTA^C and EBA variants). Figure 7.2 gives a snapshot of the situation obtained with respect to two fundamental aspects: the efficiency of the solving process (percentage of solved instances) and the CPU-time. On one hand we note that the three ESTA^C variants present a high efficiency (more than 99% of solved instances) whereas EBA finds a small set of solutions. It is important to specify that all algorithms have been executed with a time bound of 300 seconds. This explains why

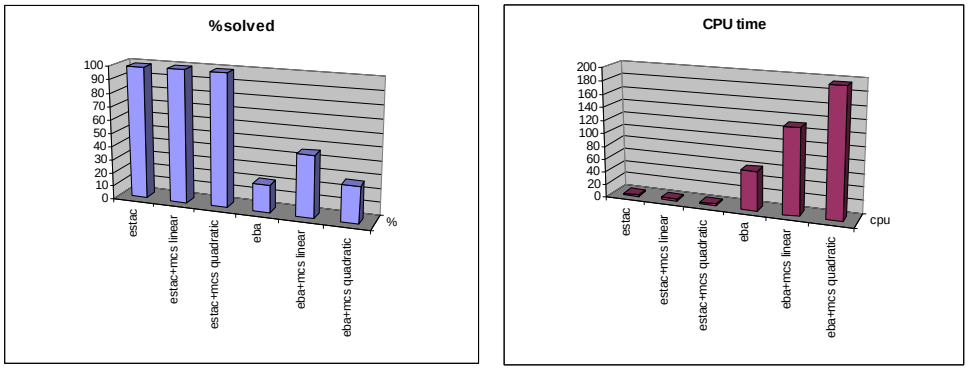


Figure 7.2: Benchmark j100. Percentage of solved instances (left-hand graph) and CPU-time in seconds (right-hand graph). All the algorithms have a time bound of 300 seconds.

	|flex|	**|fldt|**	**CPU (s)**	**npc**	**mk**
ESTA^C	0.07	0.50	0.48	68.47	374.24
ESTA^C +MCS linear	0.07	0.50	0.48	67.94	374.22
ESTA^C +MCS quadratic	0.07	0.50	0.48	68.18	374.35
EBA	0.13	0.72	33.00	53.67	501.61
EBA +MCS linear	0.13	0.58	78.64	73.41	606.31
EBA + MCS quadratic	0.11	0.56	183.79	76.88	632.27

Table 7.3: Qualities of the solutions for the subset of common solved instances of the benchmark j100.

the EBA variant that uses the quadratic MCS approach is less efficient than the EBA with linear MCS. On the other hand, considering the average CPU-time requested to solve an instance, we note that also in this case the ESTA^{C}s variants present better behaviors; in fact there are two orders of magnitude between the EBA with quadratic MCS and ESTA^C (about 2 seconds versus 200 seconds). These results show that while the solve-and-robustify approach can be applied efficiently to larger problems the envelope based analysis combines the two negative qualities of being less powerful as well as computational heavy.

Table 7.3 contains the results obtained considering the subset of solutions solved by all the algorithms. From an analysis of these data it is evident that the EBA variants produce better solutions both in terms of $flex$ and $fldt$ (also in this case the values are normalized with respect to the upper bounds obtained by relaxing the resource

constraints). Nonetheless it is worth reminding that the subset of common instances solved by all the algorithms represents less than the 20% of the overall instances. Hence it is not possible to extend these results in the evaluation of the methods. Furthermore considering the CPU time it can be noticed that for this subset of instances the ESTA^C variants require 0.48 seconds while for the whole benchmark there is an average value four times greater, 2 seconds. This underlines how such a subset is composed of "easier" instances than the average set, and emphasizes again the lack of efficency of the envelope based analisys.

To summarize, the evaluation on the benchmark j100 of the different approaches has underscored the existing gap between the envelope based analysis and the solve-and-robustify paradigm. Even though the former is able to achieve comparable results in terms of fluidity/flexibility of the solutions produced, it does not present the same power of the solving process. On the other hand, the two-step approaches exploit the use of a state-of-the-art solver, thus yielding a great percentage of solved instances and the possibility to have solutions with good makespan values.

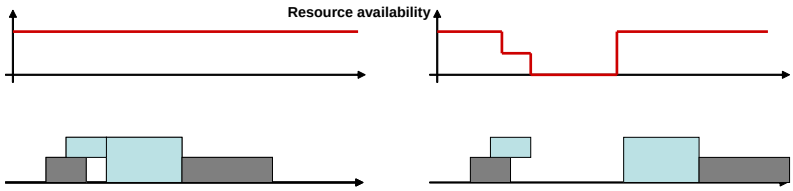
7.3 Thesis work: limitations and open issues

Robustness and schedule recovery are major issues within project and non-project scheduling. The work presented in this thesis is twofold in nature, attempting to generate better schedules with latent robustness and to perform this generation in an efficient and effective fashion. The work represents a step toward understanding the nature of robustness and schedule flexibility. There are context in which schedule generation speed is an issue and others in which it is not. The thesis addresses the cases when speed is the essence and presents the concept of partial order schedules as way to improve the responsiveness of the solver. The second part of this chapter is dedicated to providing a discussion of the issues related with the contributions of this thesis and their limitations.

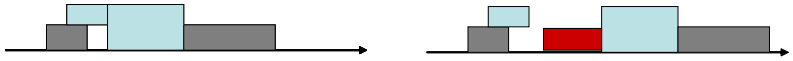
7.3.1 Partial Order Schedules and their applicability

Within the considerable body of work on scheduling, few investigations take into account execution in uncertain environments. In fact most research work is concerned with methods to obtain specific schedule qualities like minimizing the makespan or the tardiness, or maximizing possible activity rewards. Introducing the concept of Partial Order Schedules we have claimed that this can turn out to be very useful tackle the problem of scheduling uncertainty. In fact it provides a means to promptly answer to temporal changes (e.g., durations changes or delays of start-time) as well as a base to hedge against more general changes (e.g., resource capacity variations or added/deleted activities).

(a) Temporal changes: activities can last more than expected or they can be postponed until necessary conditions are satisfied.



(b) Resource changes: the two red curves represent the nominal (left) and the actual (right) resource availability. The reduction of resource availability blocks the execution of the last two activities which are delayed.



(c) Activity changes: the need to serve a new activity may create a reallocation of the current scheduled activities.

(d) Causal changes: a new precedence relation between a pair of activities can require a revision of previous choices.

Figure 7.3: Different possible changes which can be meet during the execution

In general there are different events that might disrupt the execution of the predefined schedule. In this section we analyze the bounds between which the use of $\mathcal{POS}$s gives a noticeable improvement in facing unforeseen events. A scheduling problem can be modified mainly along three different directions of uncertainty:

temporal changes : changes that involve the various temporal aspects of the problem. Among these we have:

- activities that last more than expected – change of the duration;
- delayed or anticipated activities – change of the start time;

resource changes : during the execution of a schedule it is possible to meet a reduction of the resource availability. For instance, this can happen due to the breakdown of one of the available machines;

new activities to be served : this aspect may imply the insufficiency of the available resources;

causal changes : new constraints between pairs of activities. The activities can be the ones in the original set as well as new activities to be served.

Figure 7.3 shows the different effects that the events described above can have on the solution during its execution.

Notice that using a $\mathcal{POS}$ it is possible to automatically respond to temporal and causal changes as well as to use its flexibility to repair the solution in case of resource changes. In fact in the case of temporal changes as well as causal changes, a new solution is directly obtained by synthesizing the constraint that represent the change, and then posting and propagating this constraint over the current temporal network.

In the case of resource changes it will be, in general, necessary to have a decision phase beside the propagation. The reader should notice that in the case of a new activity to serve, this might conflict, in general, with more than one of the scheduled activities. Hence, a decision phase it is necessary to select the activities to be rescheduled. We have the same in case of resource capacity variations; in fact, a reduction of capacity can be modeled by a new "dummy" activity that requires an amount of resource equal to the variation and has the start time and the duration equal respectively to the instant in which the event occurs and to its duration.

7.3.2 Methods to produce Partial Order Schedules

In the chapters 5 and 6 two broad classes of constraint-based scheduling procedures are defined and investigated: a Least Commitment and a Solve & Robustify approach. It is worth recalling that the two methods differ in the use of resource profiles to analyze the set of temporal solutions associated to the time network (see Chapter 4). The least commitment approach is based on resource profile estimation built on the whole set of solutions, whereas the second approach is based on the earliest start time profile, that is, the resource profile associated to a specific time solution - the earliest start time solution.

These two methods have been investigated because they represent the opposite ends of the spectrum. This analysis has been useful to evaluate the strengths and weaknesses of the two methods. In fact, while different investigations have highlighted the importance of more informed search based on complete resource profiles like the resource envelope ([Muscettola, 2002]), the results of this thesis show how a less informed search coupled with a robusty step obtains better results in terms of both effectiveness and efficiency. These results dampen prior claims about the expected potential of envelope based schedule generation methods.

On the other hand the reader may argue about the appropriateness of search control heuristics used for the envelope based analysis. In fact the three heuristics specified in Chapter 4 and then subsequently used within each of EBA and solve-and-robustify approaches, were in fact developed to analyze the earliest start time profiles ([Cesta *et al.*, 2002]). During this work different methods have been examined with-

out any significant result. This difficulty is still further revealed by recent work concerning the concept of resource envelope [Satish Kumar, 2003; Policella *et al.*, 2004b; Muscettola, 2004; Frank, 2004]. These, in fact, limit their analysis to increasing the quickness in computing the profile rather than to produce heuristic methods to guide the search process.

Chapter 8

Conclusions

Scheduling is a problem that occurs in a large variety of forms with huge cumulative economic and social consequences. Proper scheduling can provide better utilization of scarce and expensive resources as well as higher satisfaction for individuals such as customers and employees. There are a few reasons why scheduling is such a difficult problem:

- the size and complexity of the search space;
- scheduling is an inherently dynamic process, schedules only remain valid for a limited amount of time. After a certain duration, the world generally has changed enough that the scheduling algorithm has to find a different schedule;
- different domains and applications require solutions of different variations of the scheduling problem.

The idea of this work has been supported by the opinion that dealing with scheduling uncertainty may represent a significant step toward bridging the gap between scheduling theory and real domains. More precisely, the thesis addresses the broad question of how to build schedules that are robust in the face of a dynamic execution environment. This question is of considerable practical importance, as a major obstacle to the use of schedules in practice is their brittleness when activities do not execute as planned. Previous research has considered a number of approaches to this aspect of the scheduling problem, ranging from dynamic scheduling approaches that do not compute schedules in advance, to reactive approaches which respond to repair schedules when unexpected events force changes, to domain-specific proactive approaches that attempt to incorporate knowledge of the types of uncertainties that can arise in the target scheduling domain. In this thesis, a more generic pro-active approach is investigated, which seeks to represent, retain and exploit temporal flexibility through the use of partial-order schedules.

8.1 Contributions

The main objective of this dissertation is to study approaches to deal with uncertainty in the production of project schedules. The thesis is that flexible solutions can present better qualities to face this problem. This has led to the introduction of *Partial Order Schedules* or $\mathcal{POS}$s. Indeed, a partial order schedule on one hand allows to represent implicitly a set of solutions that can be used to follow the possible evolution of the environment during the execution, and, on the other hand, guarantees a prompt answer to such disruptive events. In the remainder of this section we list what we consider be the major contributions of this dissertation.

8.1.1 Analysis and Classification of the Current Research Scenario

In Chapter 2 different techniques based on scheduling with uncertainty have been described. We have seen that all approaches are composed of two basic ingredients: an off-line and an on-line phase. According to the ratio of these two aspects it is possible to classify the different approaches (see Fig. 2.1).

The analysis has shown how in the current state of the art there is not a single approach which emerges from the set. Rather, different results are more tailored for specific purposes. Therefore, there is a great dependency between each of the proposed approaches and the kind of scheduling problem that is taken into account. In particular in our survey we have concentrated our attention mostly on methods based on the presence of a baseline schedule. This has an important function for several reasons (see Sect. 2.4.2). The first concerns the ability of checking if there is or not the capacity to produce the planned work. Another point is that through an initial schedule it is possible to plan external activities such as material procurement and delivery of orders to customer. A baseline schedule is also vital for cash flow projections and provides a yardstick by which to measure the performance of both management and shop floor personnel.

Our analysis has established three main categories in which methods based on the construction of baseline schedules can be divided:

- Building robust schedules: the idea of these methods consists in synthesizing a solution that can tolerate a certain degree of uncertainty during its execution. In other words it should be able to absorb variations of the problem due to both external (exogenous events) and internal (false definitions in the problem) reasons.

- Rescheduling approach: in this case when during the execution a change, either external or internal, makes the current schedule invalid, these approaches proceed by recomputing a new solution. This new scheduling phase can be done both from scratch (to find optimum solutions) and locally on the sub-set of invalid activities (to guarantee a prompt answer).

- Building partial or flexible schedules: this approach consists in an intermediate approach between the use of a baseline schedule and complete dynamic approaches. This consists in computing several schedules instead of a single, brittle, one. As a result during the execution of the schedule it is possible to switch from one solution to another in case of unforeseen events or changes in the scheduling problem definition.

8.1.2 Formalization of a Flexible Approach: the Partial Order Schedule

Previous research has noted many of the potential advantages of a partial-order schedule representation from the standpoint of managing execution in an uncertain world, and has also produced some basic techniques for generating flexible schedules for specific types of scheduling problems. However, this thesis is among the first to explicitly consider the problem of how to take advantage of temporal flexibility to hedge against the dynamics of execution.

The proposed solution lies in the introduction of a partial ordering among the activities: the *Partial Order Schedule* or $\mathcal{POS}$. As described in Definition 3.1 a $\mathcal{POS}$ consists in a set of activities which are partially ordered such that any possible complete order that is consistent with the initial partial order, is a resource and time feasible schedule. More precisely a $\mathcal{POS}$ is a set of feasible schedules that can be represented by a temporal graph.

Therefore as a scheduling problem consists itself of a set of partially ordered activities, generating a $\mathcal{POS}$ requires to post further temporal constraints on the initial problem to guarantee that, once obtained a $\mathcal{POS}$, no further (temporal) constraint is able to produce a resource conflict. In other words, the solving process to obtain a partial order schedule, returns a new "problem" in which only the temporal aspect is presented while the combinatorial one has been "solved". This aspect represents a significant point. In fact, any external change or unforeseen event that can be modeled as a temporal change in the problem, can be faced promptly thanks to the characteristics $\mathcal{POS}$s. In general a partial order schedule presents the following features:

- the associated temporal graph allows to implicitly define a new solution to recover the situation after a disruptive action.

- the underlying temporal graph allows to compute the new solution quickly. In fact as said before a new solution is obtained by propagating the change over the graph and this can be accomplished by using polynomial algorithms.

- the propagation step by definition computes the minimal repairs necessary to take into account the change in input, and to obtain a new consistent solution. This minimality aspect might avoid unnecessary domino effects preserving the stability of the solution.

- the minimal repairs produced during the propagation step also allow to avoid a great "consumption" of the set of solutions represented in the $\mathcal{POS}$.

The last two points turn out to be relevant to preserve both the stability and the flexibility (or robustness) characteristics of the solution.

The necessity of posting further precedence constraints to obtain a partial order schedule has implied the use of techniques that allow to directly manage constraints during the solving process. For this reason the Constraint Satisfaction Problem paradigm has been used (Chapter 4). Furthermore Constraint Programming – that is a framework to solve problems based on the CSP representation – satisfies the need to represent the different techniques and to guide the search exploiting the knowledge of the problem.

To conclude we want to underline that, even though the characteristics of partial order schedules are tailored on temporal "change" that can be met during the execution, this kind of solution can turn out to be useful also to face other kinds of possible change like, for instance, the events which arise from resource uncertainty.

8.1.3 Flexible Solutions through Least Commitment

A crucial point of this dissertation has been the issue of how to build partial order schedules that have good robustness properties. In this regard, two broad classes of constraint-based scheduling procedures are defined and investigated. First, a novel, least-commitment approach to partial-order schedule generation, based on a recent result regarding the computation of tight upper and lower bounds on resource usage in a partial order schedule, is developed and explored. The idea behind the least commitment approach consists in reducing as much as possible the commitment implied by a decision. This results in a twofold aspect: postponing all unnecessary decisions as much as the search procedure allows it and choosing the least commitment decision once one has to be taken.

To obtain a least commitment approach we have used the resource envelope described recently in [Muscettola, 2002] that allows to compute the minimal ad maximal bounds for the resource usage considering all the possible temporal solutions admissible by a time ordering of the activities. This has produced the first scheduler that integrates the resource envelope knowledge in the solving process. The use of the resource envelope knowledge has required/allowed the study of different related aspects:

- how to boost the envelope computation: in Sect 5.3 a set of properties to reduce the complexity of computing the resource envelope have been described.

- How to manage the resource envelope to extract real conflicts: in Sect 5.4.1 we have underlined how a superficial analysis of the resource envelope can

produce aliasing effects in identifying the activities that produce resource conflicts.

- Pruning useless information through constraint propagation: the use of resource constraint propagation has been investigated to discover possible infeasible solutions and increase the efficiency of using the resource envelope.

- More intensive analysis of the search space. Finally to improve the solving process a more intensive algorithm has been implemented overcoming the possible pitfalls of heuristics choices.

In particular the method described can be viewed as an iterative repair method in which the set of all possible temporal solution of the problem is considered at each stage of the solving process. This set is then analyzed by computing the resource envelope and based on this analysis some of the temporal solutions are heuristically pruned. The process will continue until a $\mathcal{POS}$ is produced.

8.1.4 Formalization of the Solve & Robustify method

Second, a two-stage approach to partial-order schedule generation, formalized as the Solve-and-Robustify paradigm, was investigated. Under this approach, an initial fixed-time schedule is generated in the first step using earliest-start-time resource profiles, and this intermediate solution is then transformed into a partial-order schedule in the second step via a process referred to as *chaining*. Surprisingly, this solve-and-robustify approach was found to dominate the envelope-based approach in its ability to produce schedules with better robustness properties (as measured in a couple of different ways) on a set of benchmark resource-constrained project scheduling problems from the Operations Research literature. Additionally, the solve-and-robustify approach was found to solve greater numbers of problem instances, produce better makespan results, and solve problems in significantly less computation time. This result significantly dampens prior claims about the expected potential of envelope-based schedule generation techniques.

One interesting characteristic of the solve-and-robustify approach is that by definition the robustify step cannot degrade the makespan achieved by the solve step, which allows for separation of concerns and the substitution of any sort of optimizing schedule generator in the first step. Another interesting observation is that multiple chaining-form partial order schedules can be derived from a given fixed-time schedule. Pushing on this observation, the thesis next investigates the use of chaining search procedures, which utilize heuristics that exploit structural characteristics of more robust chaining form solutions and attempt to optimize with respect to different robustness measures. These iterative chaining procedures are found to produce significantly better partial order schedules with respect to various robustness measures than the basic solve-and-robustify chaining procedure. Finally, the thesis

explores additional versions of the solve-and-robustify paradigm, which interleave search for fixed-time solutions and partial order schedules to various degrees. The results here indicate a tradeoff between the generation of low-makespan and high flexibility schedules.

8.2 Future Work

Throughout this dissertation we have briefly commented on aspects of the research that remain for future work. In this section, we summarize the avenues of research suggested by the work in this dissertation.

Studying recovering techniques for non-temporal changes. The first direction we want to highlight is the possibility of exploiting the flexibility guaranteed by partial order schedules to hedge against non-temporal changes like the ones that might stem from the resource capacity reduction or the need to serve new activities.

As mentioned before a first step of a possible solution consists in noting that these two events can be represented in a common way: a new activity to be added in the scheduling problem. In fact, in the case of resource capacity reduction this can be modeled using an activity which requires an amount of resource equals to the reduction value. At this point the problem will consist in selecting where to put the new activity. This allocation will entail the modification of the allocation of other activities. For this reason selecting the right position can reduce the disruptive impact over the schedule in execution.

Using partial order schedules for integrating planning and scheduling. Architectures that integrate planning and scheduling, by interleaving the two phases might benefit from both the flexibility and the consistency that a partial order schedule can guarantee.

Even though partial order schedules cannot generally represent all the solutions that a given plan may imply, they provide a set of feasible solutions on which the planner can reason upon. In other words, a $\mathcal{POS}$ can be a common structure on which the two modules, the planner and the scheduler, negotiate.

Simulating the use of partial order schedules. Interesting results may stem from the use of partial order schedules in an empirical framework which simulates their execution. There are two main aspects which should be taken into account for the implementation of an empirical framework:

- how to simulate a “real” execution?
- how to evaluate the behavior of the execution?

Regarding the first question different points have to be taken into account. For instance, a possible approach may lie in using information about statistical distribution of disruptive events. This can be an efficient approach for well known situations. The second point can be maybe more important. In fact it is essential to have a yardstick to evaluate the quality of the execution. But it is generally hard to find a unique criterion: what is more important the stability of a solution or the preservation of the schedule quality?

Furthermore, important remarks can be made by comparing, in terms of schedule metrics, the actual execution of the solution and the best execution. This can be accomplished by solving the problem which consists of the original scheduling problem plus the various changes that arose during the execution.

8.3 Conclusion

Plans and schedules always break: tasks take longer than expected, resources are unexpectedly unavailable, products do not pass quality assessment tests and must be produced again, etc. As a result, a schedule that looks good "on paper" may quickly become irrelevant and inapplicable. As explained in this thesis, several approaches have been explored to hedge against this fact. One of the most significant (probably, the most significant) approach consists of coupling:

- a *predictive* scheduling engine, able to propose, under a compact representation, a set of possible schedules;
- with a *reactive* scheduling engine, able to interpret the represented set of schedules, and make execution decisions based on actual events.

In this approach, it is in general not necessary to call the predictive scheduling engine after each unexpected event. Actually, the predictive scheduling engine shall be called again only if a significant set of disruptions makes the represented set of schedules unusable or inappropriate in the resulting situation.

In this thesis we introduce the definition of partial order schedule as a particular set of schedules and we explicitly consider the problem of how to take advantage of temporal flexibility to hedge against the dynamics of execution. We produce several new results and insights into the issue of how to build partial order schedules. In this regard two broad classes of constraint based scheduling procedure are defined and investigated: least commitment engines attempt to generate as few (and as little constraining) delay constraints as possible, in order to solve potential resource conflicts and guarantee that all the schedules consistent with the generated delay constraints are feasible; by contrast, solve-and-robustify approaches generate first one "good" schedule, and then derive a set of delay constraints from this schedule. Surprisingly,

these solve-and-robustify approaches have been found to dominate the least commitment ones in their ability to produce schedules with better robustness properties (as measured in several different ways) on a set of benchmark resource-constrained project scheduling problems from the Operations Research literature. Additionally, the solve-and-robustify approaches have been found to solve greater numbers of problem instances, produce better makespan results, and solve problems in significantly less computation time.

Bibliography

[Ahuja *et al.*, 1993] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[Ali *et al.*, 2004] S. Ali, A. A. Maciejewski, H. J. Siegel, and J. K. Kim. Measuring the Robustness of a Resource Allocation. *IEEE Transactions on Parallel and Distributed Systems,*, 15(7):630–641, July 2004.

[Aloulou and Portmann, 2003] M. A. Aloulou and M. C. Portmann. 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)*, pages 337–362, 2003.

[Artigues and Roubellat, 2000] C. Artigues and F. Roubellat. 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, 2000.

[Artigues *et al.*, 2004] C. Artigues, J. C. Billaut, and C. Esswein. Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, 2004. To appear.

[Aytug *et al.*, 2005] H. Aytug, M. A. Lawley, K. N. McKay, S. Mohan, and R. M. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 165(1):86–110, February 2005.

[Baptiste and Le Pape, 1995] P. Baptiste and C. Le Pape. A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling. In *Proceedings of the* 14^{th} *International Joint Conference on Artificial Intelligence*, 1995.

[Baptiste *et al.*, 2001] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*, volume 39 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, 2001.

[Baptiste, 1998] P. Baptiste. *A Theoretical and Experimental Study of Resource Constraint Propagation*. PhD thesis, University of Compiègne, 1998.

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

[Bean *et al.*, 1991] J. Bean, J. Birge, J. Mittenthal, and C. Noon. Match-Up Scheduling with Multiple Resources, Release Dates and Disruptions. *Operations Research*, 39:470–483, 1991.

[Beck *et al.*, 1998] J. C. Beck, E. D. Davenport, A. J. Davis, and M. S. Fox. The ODO Project: Towards a Unified Basis for Constraint-Directed Scheduling. *Journal of Scheduling*, 1:89–125, 1998.

[Cesta and Oddi, 2001] A. Cesta and A. Oddi. Algorithms for Dynamic Management of Temporal Constraints Networks. Technical report, ISTC-CNR, Institute for Cognitive Science and Technology, Italian National Research Council, November 2001.

[Cesta and Stella, 1997] A. Cesta and C. Stella. A Time and Resource Problem for Planning Architectures. In *Proceedings of the* 4^{th} *European Conference on Planning (ECP-97)*, 1997.

[Cesta *et al.*, 1998] A. Cesta, A. Oddi, and S. F. Smith. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the* 4^{th} *International Conference on Artificial Intelligence Planning Systems, AIPS-98*, pages 214–223, 1998.

[Cesta *et al.*, 1999] A. Cesta, A. Oddi, and S. F. Smith. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the* 16^{th} *International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann, 1999.

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

[Cheng and Smith, 1994] C. Cheng and S. F. Smith. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings of the* 12^{th} *National Conference on Artificial Intelligence, AAAI-94*, pages 1086–1091. AAAI Press, 1994.

[Cormen *et al.*, 1990] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[Davenport and Beck, 2000] A. J. Davenport and J. C. Beck. A Survey of Techniques for Scheduling with Uncertainty. available on-line at http://4c.ucc.ie/ jcb/publications.html, 2000.

[Davenport *et al.*, 2001] A. J. Davenport, C. Gefflot, and J. C. Beck. Slack-based Techniques for Robust Schedules. In *Proceedings of* 6^{th} *European Conference on Planning, ECP-01*, 2001.

[Dechter and Rossi, 2002] R. Dechter and F. Rossi. Constraint Satisfaction. In L. Nadel, editor, *Encyclopedia of Cognitive Science*. Nature Publishing Group, 2002.

[Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[Demeulemeester and Herroelen, 2002] E. L. Demeulemeester and W. Herroelen. *Project Scheduling: A Research Handbook*, volume 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, 2002.

[Dijkstra, 1959] E. W. Dijkstra. A note on Two Problems in Connection with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Do and Kambhampati, 2003] M. B. Do and S. Kambhampati. Improving the Temporal Flexibility of Position Constrained Metric Temporal Plans. In *Proceedings of the* 13^{th} *International Conference on Automated Planning & Scheduling, ICAPS'03*, 2003.

[Drabble and Tate, 1994] B. Drabble and A. Tate. The Use of Optimisticand Pessimistic Resource Profiles to Inform Search in an Activity Based Planner. In *Proceedings of the* 2^{nd} *International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, 1994.

[Drummond *et al.*, 1994] M. Drummond, J. Bresina, and K. Swanson. Just-in-Case Scheduling. In *Proceedings of the* 12^{th} *National Conference on Artificial Intelligence, AAAI-94*, pages 1098–1104. AAAI Press, 1994.

[El Sakkout and Wallace, 2000] H. H. El Sakkout and M. G. Wallace. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4):359–388, 2000.

[Feige and Kilian, 1998] U. Feige and J. Kilian. Zero-knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57:187–199, 1998.

[Ford and Fulkerson, 1962] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[Fox, 1990] M. S. Fox. Constraint Guided Scheduling: A Short History of Scheduling Research at CMU. *Computers and Industry*, 14(1–3):79–88, 1990.

[Frank, 2004] J. Frank. Bounding the Resource Availability of Partially Ordered Events with Constant Resource Impact. In M. Wallace, editor, *Principles and Practice of Constraint Programming, 10^{th} International Conference, CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 242–259. Springer, 2004.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[Ginsberg *et al.*, 1998] M. L. Ginsberg, A. J. Parkes, and A. Roy. Supermodels and Robustness. In *Proceedings of the 15^{th} National Conference on Artificial Intelligence, AAAI-98*, pages 334–339. AAAI Press, 1998.

[Goldberg and Tarjan, 1988] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *Journal of ACM*, 35(4):921–940, October 1988.

[Hall and Posner, 2004] N. G. Hall and M. E. Posner. Sensitivity Analysis for Scheduling Problems. *Journal of Scheduling*, 7(1):49–83, 2004.

[Hart and Ross, 1999a] E. Hart and P. Ross. An immune system approach to scheduling in changing environments. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-99*, pages 1559–1565. Morgan Kaufman, 1999.

[Hart and Ross, 1999b] E. Hart and P. Ross. The evolution and analysis of a potential antibody library for job-shop scheduling. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 185–202. McGraw-Hill, 1999.

[Hart *et al.*, 1998] E. Hart, P. Ross, and J. Nelson. Producing Robust Schedules Via an Artificial Immune System. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.*, pages 464–469, 1998.

[Hayward *et al.*, 1989] R. Hayward, C. T. Hoang, and F. Maffray. Optimizing Weakly Triangulated Graphs. *Graphs and Combinatorics*, 5(4):339–350, 1989.

[Hebrard *et al.*, 2004a] E. Hebrard, B. Hnich, and T. Walsh. Robust Solutions for Constraint Satisfaction and Optimization. In *Proceedings of the 6^{th} European Conference on Artificial Intelligence, ECAI-04*, pages 186–190, 2004.

[Hebrard *et al.*, 2004b] E. Hebrard, B. Hnich, and T. Walsh. Super Solutions in Constraint Programming. In *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems, CP-AI-OR-04*, pages 157–172. Springer, 2004.

[Herroelen and Leus, 2004a] W. Herroelen and R. Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, July 2004.

[Herroelen and Leus, 2004b] W. Herroelen and R. Leus. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156:550–565, 2004.

[Jen, 2003] E. Jen. Stable or robust? What's the difference? *Complexity*, 8(3):12–18, 2003.

[Jensen, 2001] M. T. Jensen. Improving Robustness and Flexibility of Tardiness and Total Flow-time Job Shops using Robustness Measures. *Applied Soft Computing*, 1(1):35–52, June 2001.

[Kolisch *et al.*, 1998] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark Instances for Project Scheduling Problems. In J. Weglarz, editor, *Project Scheduling - Recent Models, Algorithms and Applications*, pages 197–212. Kluwer Academic Publishers, Boston, 1998.

[Kumar, 1992] V. Kumar. Algorithms for Constraint-Satisfaction Problems:A Survey. *Artificial Intelligence Magazine*, 13(1):32–44, 1992.

[Laborie and Ghallab, 1995] P. Laborie and M. Ghallab. Planning with Sharable Resource Constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-95*, 1995.

[Laborie, 2003] P. Laborie. Algorithms for Propagating Resource Constraints in A.I. Planning and Scheduling: Existing Approaches and New Results. *Artificial Intelligence*, 143(2):151–188, 2003.

[Leon *et al.*, 1994] V. Leon, S. D. Wu, and R. H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, September 1994.

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

[Leus, 2003] R. Leus. *The generation of stable project plans*. PhD thesis, Katholieke Universiteit Leuven, 2003.

[Mc Kay *et al.*, 1988] K. N. Mc Kay, F. R. Safayeni, and J. A. Buzacott. Job-Shop Scheduling Theory: What Is Relevant? *Interfaces*, 18:84–90, 1988.

[Mc Kay *et al.*, 2000] K. N. Mc Kay, T. E. Morton, P. Ramnath, and J. Wang. Aversion dynamics scheduling when the system changes. *Journal of Scheduling*, 3(2):71–88, 2000.

[Mehta and Uzsoy, 1998] S. V. Mehta and R. M. Uzsoy. Predictable Scheduling of a Job Shop Subject to Breakdowns. *IEEE Transactions on Robotics and Automation*, 14(3):365–378, June 1998.

[Montana *et al.*, 2000] D. Montana, J. Herrero, G. Vidaver, and G. Bidwell. A multiagent society for military transportation scheduling. *Journal of Scheduling*, 3(4):225–246, 2000.

[Montana, 2002] D. Montana. How to Make Scheduling Research Relevant. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-02*, 2002.

[Montanari, 1974] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[Morris *et al.*, 2001] P. Morris, N. Muscettola, and T. Vidal. Dynamic Control of Plans with Temporal Uncertainty. In *Proceedings of the* 17^{th} *International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[Morton and Rachamadugu, 1982] T. E. Morton and R. M. V. Rachamadugu. Myopic Heuristics for the Single Machine Weighted Tardiness Problem. Technical Report CMU-RI-TR-83-09, Robotics Institute, Carnegie Mellon University, november 1982.

[Motwani and Raghavan, 1995] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Muscettola *et al.*, 1998] N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the* 6^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 1998.

[Muscettola, 2002] N. Muscettola. Computing the Envelope for Stepwise-Constant Resource Allocations. In *Principles and Practice of Constraint Programming,* 8^{th} *International Conference, CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 139–154. Springer, 2002.

[Muscettola, 2004] N. Muscettola. Incremental Maximum Flows for Fast Envelope Computation. In *Proceedings of the* 14^{th} *International Conference on Automated Planning & Scheduling, ICAPS'04*, 2004.

[Neumann and Schwindt, 1999] K. Neumann and C. Schwindt. Project Scheduling with Inventory Constraints. Technical Report WIOR-572, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, 1999.

[Nuijten and Aarts, 1996] W. P. M. Nuijten and E. H. L. Aarts. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

[Nuijten and Le Pape, 1998] W. Nuijten and C. Le Pape. Constraint-Based Job Shop Scheduling with Ilog-Scheduler. *Journal of Heuristics*, 3(4):271–286, March 1998.

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

[Ow *et al.*, 1988] P. S. Ow, S. F. Smith, and R. E. Howie. CSS: A Cooperative Scheduling System. In M. D. Oliff, editor, *Expert Systems and Intelligent Manufacturing*. Elsevier Science Publishing, 1988.

[Policella *et al.*, 2003] N. Policella, S. F. Smith, A. Cesta, and A. Oddi. Steps toward computing flexible schedules. In *Proceedings of Online-2003 Workshop CP 2003*, 2003.

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

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

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

[Roy and Sussman, 1964] B. Roy and B. Sussman. Les problemes d'ordonnancement avec contraintes disjonctives. Note DS n. 9 bis, SEMA, Paris, 1964.

[Sadeh, 1991] N. M. Sadeh. *Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh PA, March 1991.

[Satish Kumar, 2003] T. K. Satish Kumar. Incremental computation of resource-envelopes in producer consumer models. In *Principles and Practice of Constraint Programming, 9^{th} International Conference, CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 664–678. Springer, 2003.

[Schäffter, 1997] M. Schäffter. Scheduling with respect to forbidden sets. *Discrete Applied Mathematics*, 72:141–154, 1997.

[Schwindt, 1998] C. Schwindt. A Branch and Bound Algorithm for the Resource-Constrained Project Duration Problem Subject to Temporal Constraints. Technical Report WIOR-544, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, 1998.

[Sevaux and Sörensen, 2002] M. Sevaux and K. Sörensen. A genetic algorithm for robust schedules in a just-in-time environment. Technical Report LAMIH/SP-2003-1, University of Valenciennes, 2002.

[Smith and Cheng, 1993] S. F. Smith and C. Cheng. Slack-based Heuristics for Constraint Satisfactions Scheduling. In *Proceedings of the* 11^{th} *National Conference on Artificial Intelligence, AAAI-93*, pages 139–144. AAAI Press, 1993.

[Smith, 1994a] S. F. Smith. OPIS: A Methodology and Architecture for Reactive Scheduling. In M. Fox and M. Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[Smith, 1994b] S. F. Smith. Reactive Scheduling Systems. In D. Brown and W. Scherer, editors, *Intelligent Scheduling Systems*. Kluwer Academic Publishers, 1994.

[Smith, 2003] S. F. Smith. Is Scheduling a Solved Problem? In *Proceedings First Multi-Disciplinary International Conference on Scheduling: Theory and Applications (MISTA 03)*, 2003. Invited Keynote Talk.

[Sotskov, 1991] Y. N. Sotskov. Stability of an optimal schedule. *European Journal of Operations Research*, 55(1):91–102, 1991.

[Stork, 2001] F. Stork. *Stochastic Resource-Constrained Project Scheduling*. PhD thesis, Technische Universität Berlin, 2001.

[Sycara *et al.*, 1991] K. P. Sycara, S. F. Roth, N. Sadeh, and M. S. Fox. Resource Allocation in Distributed Factory Scheduling. *IEEE Expert*, 6(1):29–40, 1991.

[Tavares *et al.*, 1998] L. V. Tavares, J. A. Ferreira, and J. S. Coelho. On the optimal management of project risk. *European Journal of Operational Research*, 107(2):451–469, 1998.

[Tsamardinos *et al.*, 1998] I. Tsamardinos, N. Muscettola, and P. Morris. Fast transformation of temporal plans for efficient execution. In *Proceedings of the* 15^{th} *National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[Tsang, 1993] E. P. K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

[Vidal and Fargier, 1997] T. Vidal and H. Fargier. Contingent durations in temporal csps: from consistency to controllabilities. In *Proceedings of IEEE TIME-97 International Workshop*, 1997.

[Vidal and Ghallab, 1996] T. Vidal and M. Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings of the 12^{th} European Conference on Artificial Intelligence*, pages 48–52, 1996.

[Vieira *et al.*, 2003] G. E. Vieira, J. W. Herrmann, and E. Lin. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

[Wallace, 2000] S. W. Wallace. Decision making under uncertainty: is sensitivity analysis of any use? *Operations Research*, 48(1):20–25, 2000.

[Weigel and Bliek, 1998] R. Weigel and C. Bliek. On Reformulation of Constraint Satisfaction Problems. In *13^{th} European Conference on Artificial Intelligence, ECAI98*, pages 254–258, Brighton, UK, 1998.

[Weiss, 1999] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999.

[Wu *et al.*, 1999] S. D. Wu, E. S. Beyon, and R. H. Storer. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, January – February 1999.

[Zweben *et al.*, 1994] M. Zweben, E. Davis, B. Daun, and M. J. Deale. Scheduling and Rescheduling with Iterative Repairs. In M. Fox and M. Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.