Chapter 6
A Precedence Constraint Posting Approach

Amedeo Cesta, Angelo Oddi, Nicola Policella, and Stephen F. Smith

Abstract This chapter summarizes some previous work on a constraint-based scheduling approach effectively applied to Resource-Constrained Project Scheduling problems. The approach is based on a formulation of the problem as a Constraint Satisfaction Problem (CSP). In particular the problem is reduced to the one of establishing sufficient precedence constraints between activities that require the same resource so as to eliminate all possible resource contention, defining what is called the *Precedence Constraint Posting* (PCP) approach. The PCP scheduling approach has two attractive properties: first it operates in a search space that avoids over-commitment to specific activity start times, and can be more efficiently searched; second, the solution generated is a so-called "flexible schedule", designating a set of acceptable futures, which provides a basis for efficiently responding to unexpected disruptions during execution. This chapter summarizes a body of work developed over the years on PCP-based scheduling to take advantage of such properties. In particular, the chapter presents an overview on a number of original algorithms for efficiently finding a solution to a scheduling problem, for generating robust schedules, and for searching near-optimal makespan solutions.

Keywords Constraint-based reasoning • Generalized precedence relations • Makespan minimization • Renewable resources • Robust scheduling • Temporal flexibility

A. Cesta (✉) • A. Oddi
Institute of Cognitive Sciences and Technologies, CNR - Italian National Research Council, Rome, Italy
e-mail: amedeo.cesta@istc.cnr.it; angelo.oddi@istc.cnr.it

N. Policella
European Space Operations Centre, European Space Agency, Darmstadt, Germany
e-mail: nicola.policella@esa.int

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

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_6

6.1 Introduction

Constraint-based scheduling models combine rich representational flexibility with compositional search procedures and heuristics, and this combination can provide significant leverage in addressing complex scheduling problems. These approaches start with a formulation of the problem of interest as a *Constraint Satisfaction Problem* (CSP), which involves specification of a set of decision variables together with a set of constraints on their mutual values. Much like the MILP formulations discussed in Chap. 2 of this handbook, there are different possibilities. One basic approach is to formulate the problem as one of finding a consistent assignment of start times for all constituent activities to be scheduled (analogous to the time-indexed formulations of Chap. 2 of this handbook). This fixed times assignment formulation, however, has a couple of drawbacks. From a constraint reasoning perspective, it promotes over-commitment (i.e., the problem constraints likely do not require commitment to specific start times to ensure feasibility), which results in a correspondingly larger solution space to search. From an operational perspective, a fixed-times solution offers no resilience against executional uncertainty, and is likely to quickly become invalid.

An alternative CSP formulation, which we adopt in this chapter, is to reduce the problem to one of establishing sufficient precedence constraints between activities competing for the same resources to eliminate all possible resource contention, and ensure both time and resource feasibility. This approach, referred to as Precedence Constraint Posting (PCP), address the over-commitment issue raised above by instead producing a so-called "flexible schedule" where activity start times are constrained to occur within an interval that is consistent with the problem constraints. The original formulation (Smith and Cheng 1993) was shown to achieve order-of-magnitude speedup in solving time over a corresponding fixed times assignment procedure in a basic job shop scheduling setting. More recently, this approach has been generalized to address cumulative resources, has been effectively applied to the resource constrained project scheduling problem (Cesta et al. 2002), and has been extended to generate Partial Order Schedules (POSs), activity networks with the property that any temporal solution to the graph is also resource-feasible (Policella et al. 2007). The ability to generate POSs provides a basis for efficiently responding to unexpected disruptions at execution time. It also provides a conceptual framework for optimizing solution robustness in the absence of knowledge about the uncertainties in the execution environment (similar in some ways to the robust scheduling techniques discussed in Chap. 40 in the second volume of this handbook, and in contrast to the stochastic models of Chap. 37 in the second volume of this handbook).

In this chapter we summarize this PCP approach to Project Scheduling. We present a basic set of core solving algorithms for generating a solution in the form of an activity network N, a directed graph $N = (V, E)$, where the edges in E are simple precedence constraints imposed on the set of problem activities V. Via a polynomial time calculation, such a network N can be always turned into a Partial

Order Schedule (*POS*). We also discuss incorporation of these core algorithms into extended optimizing search procedures targeted on two different objectives: improve the robustness of a solution (Policella et al. 2009) and minimize the solution makespan (e.g., Oddi et al. 2010a).

The chapter is organized as it follows: Sect. 6.2 reviews the state-of-the-art in Constraint-based Scheduling. After the introduction of the reference scheduling problem (RCPSP/max, Sect. 6.3) and its CSP representation (Sect. 6.4), the chapter than continues describing the basic concepts behind PCP (Sect. 6.5). A core PCP framework is discussed in Sect. 6.6. The last part of the chapter then summarizes the results along two different directions of work: a first for generating robust schedules (Sect. 6.7); a second one for the generation of optimal solutions (Sect. 6.8). The final conclusions are discussed in Sect. 6.9.

6.2 Constraint-Based Scheduling

Constraint Programming (see Rossi et al. 2006) is an approach to solving combinatorial search problems based on the Constraint Satisfaction Problem (CSP) paradigm (Kumar 1992; Montanari 1974; Tsang 1993). Constraints are just relations and a CSP states which relations should hold among the given problem *decision variables*. This framework is based on the combination of *search* techniques and *constraint propagation*. Constraint propagation consists of using constraints actively to prune the search space. Different propagation 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. The filtering algorithm is invoked any time a domain of some variable is changed (either as a result of search decisions or during constraint propagation) to propagate the consequences of the change over all decision variables. As described by Dechter and Rossi (2002), "in general, constraint satisfaction tasks, like finding one or all solutions or the best solution, are computationally intractable, $\mathcal{NP}$-hard". For this reason the constraint propagation process cannot be complete, that is, some infeasible values may still remain in the domains of the variables and thus decisions are necessary to find a complete feasible valuation of the variables. In general, constraint propagation is aimed at achieving local consistency among subsets of variables.

Constraint satisfaction and propagation rules have been successfully used to model, reason and solve about many classes of problems in such diverse areas as scheduling, temporal reasoning, resource allocation, network optimization and graphical interfaces. In particular, CSP approaches have proven to be an effective way to model and solve complex scheduling problems (see, for instance, Baptiste et al. 2001; Beck et al. 1998; Cesta et al. 2002; Fox 1990; Sadeh 1991; Smith 1994). 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.

In the remainder of this section we first provide a formal definition of the Constraint Satisfaction Problem, next a brief description of a generic solver, and finally an overview of the different ways in which this paradigm has been used to solve scheduling problems.

6.2.1 Constraint Satisfaction Problem

A *Constraint Satisfaction Problem*, CSP, consists of a finite set of *decision 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. Therefore, a CSP is defined by the tuple $\langle V, D, \mathfrak{C} \rangle$ where:

- $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n variables;
- $D = \{D_1, D_2, \ldots, D_n\}$ is the set of corresponding domains of values for any variable, such that $v_i \in D_i, i = 1, \ldots, n$;
- $\mathfrak{C} = \{c_1, c_2, \ldots, c_\nu\}$, is a set of ν constraints, $c_\mu(v_1, v_2, \ldots, v_n)$, that are predicates defined on the Cartesian product of the variable domains, $D_1 \times D_2 \times \ldots \times D_n$.

A *solution* s is a value assignment to each variable v_i, 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). In this case an *objective function* $f(S)$ (or cost function) evaluates any single feasible solution S. The goal is to find an optimal solution S^* which minimizes (or maximizes) the objective function $f()$.

Finally, we observe an instance of a CSP $\langle V, D, \mathfrak{C} \rangle$ can be represented as a constraint graph, $G = (V, E)$. For every variable $v_i \in V$, there is a corresponding node in the graph. For every set of variables connected by a constraint $c_j \in \mathfrak{C}$, there is a corresponding hyper-edge. In the particular case of binary constraints (each constraint involves at most two variables) the hyper-edges become simply edges. A well known example of binary CSP (extensively used in this chapter) is the Simple Temporal Problem (STP) introduced by Dechter et al. (1991).

6.2.2 A Generic CSP Solver

A complete CSP solving procedure consists of the following three steps (Algorithm 6.1): (a) current problem P is checked for consistency [CheckConsistency(P)] by the application of a *propagation* procedure, if at least one constraint is violated the algorithm exits with failure. If the problem P is also a solution [IsSolution(P)],

Algorithm 6.1: CSP-Solver(P)

```
if CheckConsistency(P) then
    if IsSolution(P) then
        S ← P
        return S
    else
        v_i ← SelectVariable(P)
        λ_i ← Choose-Value(P, v_i)
        CSP-Solver(P ∪ {λ_i})
    end if
else
    return ∅
end if
```

then the algorithm exits and returns the generated solution $S = P$. Otherwise the problem is still not solved and the following two steps are executed; (b) a variable v_i is selected by a *variable ordering* heuristic; (c) a value λ_i is chosen by a value ordering heuristic and added to P. The solver is recursively called on the updated problem $P \cup \{\lambda_i\}$.

6.2.3 CSP Approaches to Scheduling Problems

Scheduling problems are difficult combinatorial optimization problems and represent an important application area for constraint directed search. Different constraint programming approaches have been developed in this direction, for instance, the reader can refer to Baptiste et al. (2001) for a thorough analysis of different constraint based techniques for scheduling problems. The work of Constraint directed Scheduling of the 1980s (see for example Fox 1990; Sadeh 1991; Smith 1994) has developed into Constraint-based Scheduling approaches in the late 1990s (see Baptiste and Le Pape 1995; Beck et al. 1998; Hentenryck and Michel 2009; Nuijten and Aarts 1996; Smith and Pyle 2004). These approaches all focus on the use of constraints as a basis for representing and managing the search for a solution to the scheduling problem at hand. As mentioned above, the search for a solution to a CSP can be viewed as modifying the constraint graph $G = (V, E)$ through the addition and removal of constraints, where the constraint graph is an evolving representation of the search state, and a solution is a state in which a single value remains in the domain of each variable and all constraints are satisfied.

As mentioned at the outset, research in constraint-based scheduling has pursued two basic formulations of the scheduling problem. One set of approaches (e.g., Nuijten and Le Pape 1998; Sadeh 1991; Smith and Pyle 2004) has formulated the problem as that of finding a consistent assignment of start times for each activity to be performed. Under this model, decision variables are time points that designate the start times of various activities and CSP search focuses on determining a consistent

assignment of start time values. The "serial and parallel" methods discussed in Chap. 1 of this handbook similarly aim to find consistent sets of start times. A second set of approaches have focused on a problem formulation more akin to least-commitment frameworks. In this model, which is based on a disjunctive graph representation (Adams et al. 1988) and is referred to as *Precedence Constraint Posting* (Smith and Cheng 1993), solving consists of posting additional precedence constraints between pairs of activities contending for the same resources to ensure feasibility with respect to time and capacity constraints. 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.

6.3 The Reference Scheduling Problem: RCPSP/max

We adopt the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max,[1] as a reference problem (see Bartusch et al. 1988). The basic entities of interest in this problem are *activities*. The set of activities is denoted by $V = \{1, 2, \ldots n\}$ where each activity has a fixed *processing time*, or *duration*, p_i and must be scheduled without *preemption*.

A *schedule* is an assignment of start times to each activity in V, i.e., a vector $S = (S_1, S_2, \ldots, S_n)$ where S_i denotes the start time of activity i. The time at which activity i has been completely processed is called its *completion time* and is denoted by C_i. Since we assume that processing times are deterministic and preemption is not permitted, completion times are determined by:

$$C_i = S_i + p_i \quad (i \in V) \tag{6.1}$$

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,

$$d_{ij}^{min} \leq S_j - S_i \leq d_{ij}^{max} \quad ((i, j) \in V) \tag{6.2}$$

where d_{ij}^{min} and d_{ij}^{max} are the minimum and maximum time lag of activity j relative to i. A schedule $S = (S_1, S_2, \ldots, S_n)$ is *time feasible*, if all inequalities given by the activity precedences/time lags (6.2) and durations (6.1) hold for start times S_i.

During their processing, activities require specific resource units from a set $\mathscr{R} = \{1, 2, \ldots, K\}$ of resources. Resources are *reusable* (renewable), i.e., they are released when no longer required by an activity and are then available for use by another activity. Each activity $i \in V$ requires r_{ik} units of the resource $k \in \mathscr{R}$ during its processing time p_i. Each resource $k \in \mathscr{R}$ has a limited capacity of R_k units.

[1] The three field classification of RCPSP/max is $PS|temp|C_{max}$.

A schedule S is *resource feasible* if at each time t the demand for each resource $k \in \mathscr{R}$ does not exceed its capacity R_k, i.e.,

$$r_k(S,t) = \sum_{i \in V : S_i \leq t < C_i} r_{ik} \leq R_k \quad (k \in \mathscr{R};\ t \in [0,T]) \tag{6.3}$$

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

The RCPSP/max problem is a complex scheduling problem: in fact not only the optimization version but also the feasibility problem is $\mathscr{NP}$-hard (see Bartusch et al. 1988). The reason for this $\mathscr{NP}$-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.

6.4 The RCPSP/max as a CSP

As introduced above, we formulate the project scheduling problem as a Constraint Satisfaction Problem (CSP), in particular we refer to the definition of the problem proposed in the work Bartusch et al. (1988), such that *decision variables* are the so-called *Forbidden Sets* also known as *Minimal Critical Sets* (see Laborie and Ghallab 1995 or Chap. 2 of this handbook, we use MCS in the rest of the chapter).

Given a generic resource k, a *conflict* is a set of activities requiring the resource k, which can mutually overlap and whose combined resource requirement is in excess of the resource capacity R_k. A Minimal Critical Set, $MCS \subseteq V$, represents a resource conflict of minimal size (each subsets is not a resource conflict), which can be *resolved* by posting a single precedence constraint between two of the competing activities in the conflict set. Hence, in CSP terms, a decision variable is defined for each *MCS* and the domain of possible vales is the set of all possible feasible precedence constraints $i \prec j$ which can be imposed between any pair of activities in the MCS.

A *solution* of the scheduling problem is a set of precedence constraints (added to the original problem described in the previous Sect. 6.3) such that removes all the MCSs.

A solution takes the form of an activity network N_S, a directed graph $N_S = (V_S, E)$, where $V_S = V \cup \{source, sink\}$, the set of problem activities V plus two fictitious activities *source* and *sink*, and E is the set of directed edges (i,j), representing the set of precedence constraints $i \prec j$ defined among the activities in V_S. In particular, the set E is partitioned in two subsets, $E = E_{prob} \cup E_{post}$, where E_{prob} is the set of precedence constraints originating from the problem definition and E_{post} is the set of precedence constraints posted to resolve resource conflicts. In general, the directed graph $N_S(V_S, E)$ represents a set of temporal solutions $(S_1, S_2, \ldots, S_n)$, that is a set of assignments to the activities' start-times which are consistent with the set of constraints E and the set of imposed resource constraints.

We observe as in the new formulation of the problem it becomes a *pure* disjunctive temporal problem (Oddi et al. 2010b), such that the original resource constraints are *compiled* into a set of MCSs, each MCS can be seen as a disjunctive temporal clause. A drawback of the previous MCS reduction is the large number of MCSs obtained for each resource k; if n_k is the number activities requiring resource k, the number of MCSs is $\binom{|n_k|}{R_k+1}$, hence $\mathcal{O}(n_k^{R_k+1})$.

The next section proposes a summary of the works (Cesta et al. 1999, 2002) which describes how to overcame the limitation imposed by the large size set of MCSs. The proposed approach, targeted to identification of decision variables, attempts to reconcile two, typically conflicting desiderata: (1) on one hand to always take the decision centers on the most critical precedence constraint to post and (2) on the other to minimize the amount of time spent in the analysis that leads to this decision. For details on the empirical evaluation of the algorithms the author can refer to the original works (Cesta et al. 1999, 2002).

6.5 Precedence Constraint Posting

The proposed Precedence Constraint Posting (PCP) approach was first introduced by Smith and Cheng (1993) for problems with binary resources and then extended to more general problems in subsequent research, aims 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 this approach is provided in Fig. 6.1 and consists of representing, analyzing, and solving different aspects of the problem in *two separate layers*. In the former the temporal aspects of the scheduling problem, e.g., activity durations, constraints between pairs of activities, due dates, release time, etc., are considered. The second layer, instead, represents and analyzes the resource aspects of the problem. Let us now explain the details of the two layers.

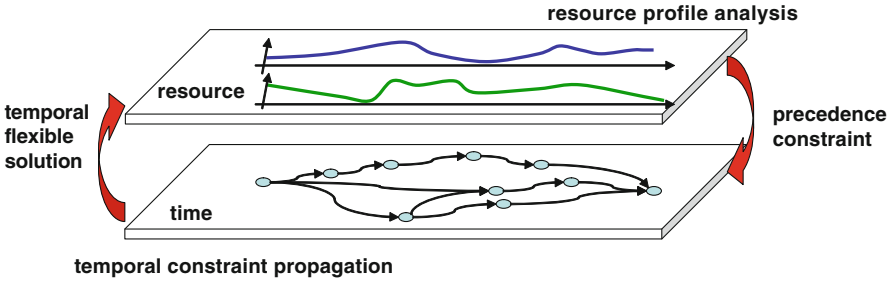

Fig. 6.1 Precedence Constraint Posting schema

6.5.1 Time Layer

The temporal aspects of the scheduling problems are represented through an STP (Simple Temporal Problem) network (see Dechter et al. 1991). This is a temporal graph in which the set of nodes represents a set of temporal variables named *time-points*, tp_μ, while linear temporal constraints, of the form $tp_\mu - tp_\nu \leq d_{\mu\nu}$, define the distances among them. Each time point has initially a domain of possible values equal to $[0, T]$ where T is the temporal horizon of the problem. The problem is represented by associating with 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 imposed between a pair of time points that can "belong" to the same activity or not. In the latter case (when they do not belong to the same activity) the temporal constraints represent constraints between two activities of the problem. If alternatively, the two time-points belong to the same activity, the temporal constraints represent the duration, or processing time, of the activity. By propagating the temporal constraints it is possible to bound the domains of each time-point, $tp_\mu \in [lb_\mu, ub_\mu]$. In the case of empty domains for one or more time-points the temporal graph does not admit any solution. In Dechter et al. (1991) it was proved that it is possible to completely propagate the whole set of temporal constraints in polynomial time, $\mathcal{O}(n^3)$, and, moreover, that a solution can be obtained by selecting for each time-point its lower bound value, $tp_\mu = lb_\mu$ (this solution is referred to as the *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 need to also be proven to be resource feasible.

6.5.2 Resource Layer

This layer takes into account the other aspect of the scheduling problem, namely the constraints on resources (i.e., capacity). In general, resources can be binary, multi-capacitive, or consumable (non-renewable). As described above, the input to this layer in the PCP approach is a temporally flexible solution—a set of temporal solutions (see also Fig. 6.1). Like in the previous layer it is possible to use constraint propagation to reduce the search space. Even though there are different methodologies described in the literature, these propagation procedures are not sufficient in general (see Laborie 2003; Nuijten and Aarts 1996). In fact they are not complete, which implies that they are not able to prune all inconsistent temporal solutions. For this reason a PCP procedure uses a *Resource Profile* (see Cesta et al. 2002) to analyse resource usage over time and detect MCS decision variables. The procedure then proceeds to post further constraints to level (or solve) some of the detected conflicts. 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 again using the resource profiles. The search stops when either the temporal graph becomes inconsistent or the resource profiles are consistent with the resource capacities.

6.6 The Core Constraint-Based Scheduling Framework

The core of the implemented framework is based on the greedy procedure described in Algorithm 6.2, which is an instance of the procedure described in Sect. 6.2.2. Within this framework, a solution is generated by progressively detecting time periods where resource demand is higher than resource capacity (conflicts) 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, $S^0 \leftarrow P$, the procedure builds an estimate of the required resource profile according to the current temporal precedences in the network and detects resource conflicts—SELECTCONFLICTSET(S^0). If the set of conflicts F is not empty then new constraints are synthesized, SELECTLEVELING-CONSTRAINT(F), and posted on the current situation. The search proceeds until either the STP temporal graph becomes inconsistent or a solution is found.

Using the reference scheduling problem (RCPSP/max) introduced above, we proceed now to summarize the core components needed to fully specify the approach. In fact, the introduction of a specific scheduling problem allows us to set the general framework introduced above. The first issue in the framework implementation concerns how to identify activities that are in a conflicting situation. This allows identification of the points in the current solution state that need to be resolved. The second issue concerns the heuristics (variable and value ordering) used to respectively select and solve conflicts that have been identified.

6.6.1 Consider Resource Utilization

A first, important issue that needs to be explored is how to compute *resource utilization 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 of computing bounds on resource utilization. Examples of bounding procedures can be found in Drabble and Tate (1994), Cesta and Stella (1997), Laborie (2003), Muscettola (2002). It is worth noting that consideration of resource bounds as resource profiles implies that all temporal solutions represented by the temporal graph are also resource feasible.

A different approach to dealing with resources consists of focusing attention on a specific temporal solution and its resource utilization. In contrast to resource

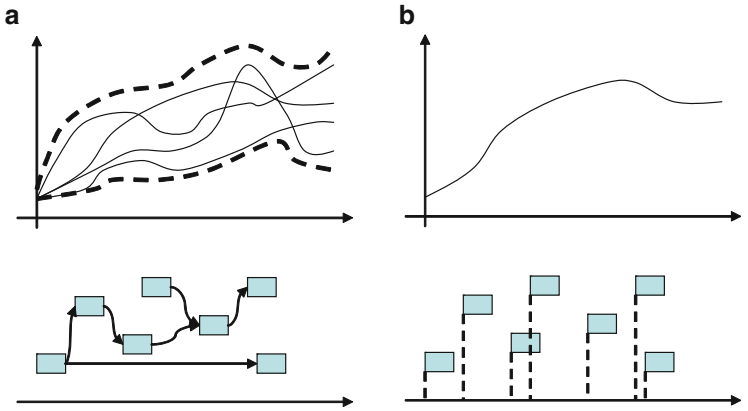

Fig. 6.2 Two different ways to consider the resource utilization. (**a**) Bounds of the resource utilization for the set of solutions defined by a temporal graph. (**b**) Resource utilization of a single temporal solution

bounding approaches, this process only assures that the final temporal graph contains at least one resource feasible solution (the one for which the resource utilization is in fact considered); some of the temporal solutions may not be resource feasible. Since only a single feasible solution is computed instead of encapsulating all possible feasible solutions, this approach gains substantial computational efficiency.

Figure 6.2 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. 6.2a). Alternatively, only one temporal solution of the set is considered in the second case (Fig. 6.2b).

6.6.2 How to Identify Decision Variables

The starting point in identifying the possible conflicts is the computation of the possible contention *peaks*. A *contention peak* is 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). In Algorithm 6.2 the function SELECTCONFLICTSET(S^0) collects all *maximal peaks*[2] in the current schedule. Then selects a decision variable (MCS) from the set of peaks. An alternative selection procedure first ranks the selected peaks, next picks the more critical one (e.g., one with maximal size), and last selects a decision

[2]Specifically, we follow a strategy of collecting sets of activities such that none of the sets is a subset of the others.

Algorithm 6.2: GREEDYPCP(P)

Require: a problem P
Ensure: a solution S (or the empty set otherwise)
  $S^0 \leftarrow P$
  if Exists an unresolvable conflict in S^0 **then**
    $S \leftarrow \emptyset$
  else
    $F \leftarrow$ SELECTCONFLICTSET(S^0)
    if $F = \emptyset$ **then**
      $S \leftarrow S^0$
    else
      $\{i \prec j\} \leftarrow$ SELECTLEVELINGCONSTRAINT(F)
      $S^0 \leftarrow S^0 \cup \{i \prec j\}$
      $S \leftarrow$ GREEDYPCP(S^0)
    end if
  end if
  return S

variable from the selected peaks. The selected MCS is solved by imposing on the conflicting activities a single precedence constraint $i \prec j$. Next Sect. 6.6.3 gives more details about the used selection procedures.

Cesta et al. (1999, 2002) showed that much of the advantage of this type of global conflict analysis can be retained by using an approximate polynomial procedure for computing MCSs. In particular, Cesta et al. (1999, 2002) proposed two polynomial strategies for sampling MCSs from a peak of size $|F|$. These strategies are based on the idea of sorting the activities of each peak according to their resource usage (greatest first), then MCSs are collected by visiting such a list and extracting subsequences of activities corresponding to MCSs. The two methods are named *linear* and *quadratic* according to their complexity (they respectively collect $\mathcal{O}(|F|)$ and $\mathcal{O}(|F|^2)$ elements). An additional and effective strategy is based on the idea of imposing a *lexicographical* order on the set of searched MCSs, the reader can refer to the paper Cesta et al. (2002) to see the detail of the procedure.

6.6.3 Selecting and Solving Conflicts

According to the proposed CSP framework, in all cases where no mandatory decisions can be deduced from the propagation phase, heuristics and methods used to respectively select and solve one of the conflicts are introduced by defining *variable* and *value* ordering heuristics for the decision variables. The basic idea is to repeatedly evaluate the decision variables and select the one with the best heuristic evaluation. The selection of which variable to assign next is based on the *most constrained first* (MCF) principle, and the selection of values follows the *least constraining value* (LCV) heuristic. More specifically, the following heuristics are assumed:

Ranking conflicts: for evaluating MCSs we have used the heuristic estimator $\kappa()$ described by Laborie and Ghallab (1995), where the MCS with highest value of $\kappa(MCS)$ is then chosen. A conflict is unsolvable if no pair of activities in the conflict can be ordered. Basically, $\kappa()$ will measure how close a given conflict is to being unsolvable.

Slack-based value selection: to choose an ordering decision among the possible, 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.

6.7 Flexible Solutions, Robustness, and Partial Order Schedules

As shown in the previous section, the outcome of a Precedence Constraint Posting solver is a Simple Temporal Problem (STP) network or STN, that not only contains the temporal constraints belonging to the initial problem, but also the additional precedences that have been added during the resolution process. In a STN, each time point is associated with a bounded interval of values which represents the set of admissible values for that time point; hence, the adjective "temporally flexible" is often used to refer to these kind of solutions. Therefore the PCP approach tries to retain the temporal flexibility of the underlying STN to the extent possible (somehow maximizing the domain size of the time points). In particular, the use of PCP approaches (and the schedules produced by them) can be justified in two ways:

- As a means of retaining the flexibility implied by problem constraints (time and capacity) and avoiding over commitment;
- As a means of establishing conditions for guaranteed *executability*.

In fact, in most practical scheduling environments, off-line schedules can have a very limited lifetime and scheduling is really an ongoing process of responding to unexpected and evolving circumstances. In such environments, insurance of robust response is generally the first concern. Unfortunately, the lack of guidance that might be provided by a schedule often leads to myopic, sub-optimal decision-making.

One way to address this problem is *reactively*, through schedule repair. To keep pace with execution, the repair process must be both fast and complete. The response to a disruption must be fast because of the need to re-start execution of the schedule as soon as possible. A repair must also be complete in the sense of accounting

for all changes that have occurred, while attempting to avoid the introduction of new changes. As these two goals can be conflicting, a compromise solution is often required. Different approaches exist and they tend to favour either timeliness (Smith 1994) or completeness (El Sakkout and Wallace 2000) of the reactive response.

An alternative, *proactive* approach to managing execution in dynamic environments is to focus on building schedules that retain flexibility and are able to absorb some amount of unexpected events without rescheduling. One technique consists of factoring time and/or resource redundancy into the schedule, taking into account the types and nature of uncertainty present in the target domain (Davenport et al. 2001; Hiatt et al. 2009). An alternative technique is to construct an explicit set of contingencies (i.e., a set of complementary solutions) and use the most suitable with respect to the actual evolution of the environment (Drummond et al. 1994). Both of these proactive techniques presume an awareness of the possible events that can occur in the operating environment, and in some cases, these knowledge requirements can present a barrier to their use.

Research approaches are based on different interpretations of the concept of a robust solution, e.g., the ability to preserve solution qualities or the ability to maintain a stable solution. The concept of robustness, on which this work is based, 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.

Figure 6.3a 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

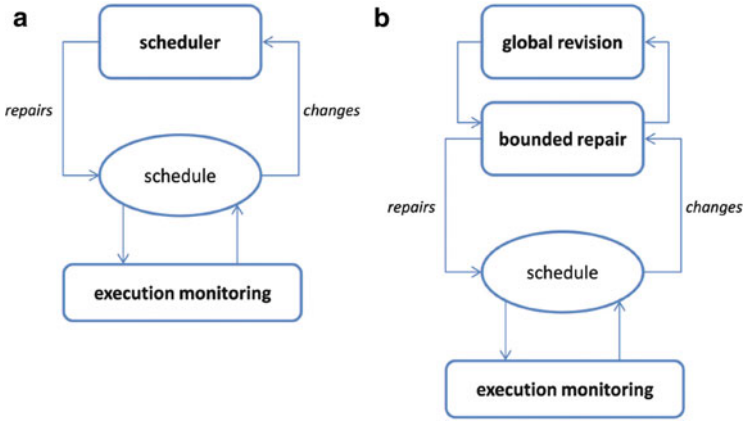

Fig. 6.3 Rescheduling actions during the execution. (**a**) General rescheduling phase. (**b**) Rescheduling phase using a flexible solution

solution is computed, it is given back to the executor. In Fig. 6.3b, instead, the execution of a flexible schedule is highlighted. The substantial difference in this case is that the use of flexible solutions allows the introduction of two separate rescheduling phases: the first enabling rapid response by immediate means like temporal constraint propagation over the set of activities, and the second entailing to more extensive re-computation of the schedule when the first phase cannot offer a response. In practice, the first phase exploits the flexibility characteristics of the solution (and for this reason we named this module *bounded repair scheduler*). Of course it is possible that an unforeseen event will force the system outside of the bounds provided by the flexible solution. In this case, it will be necessary to invoke the second, more complete scheduling phase. This second phase involves re-computation of the overall flexible schedule, and performs a much more extensive constraint-based search procedure (*global revision* module). Note that the use of flexible schedules makes it possible to bypass this extended computation in many circumstances in favor of a prompt answer.[3]

6.7.1 Partial Order Schedules

To take maximum advantage of the opportunity to bypass extended computation in the event of unexpected events, a stronger form of flexible schedule is required. Policella et al. (2004) and Policella (2005) further elaborated the idea of exploiting temporal flexibility by adopting a graph formulation of the scheduling problem and focusing on generation of the *Partial Order Schedules* (*POS*s).

Definition 6.1 (Partial Order Schedule). A Partial Order Schedule *POS* for a problem P is an activity network, such that any possible temporal solution is also a resource-consistent assignment.

Within a *POS*, each activity retains a set of feasible start times, and these options provide a basis for responding to unexpected disruptions.

An attractive property of a *POS* is that reactive response to many external changes can be accomplished via simple propagation in an underlying temporal network (a polynomial time calculation); only when an external change exhausts all options for an activity is it necessary to recompute a new schedule from scratch. 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 to the temporal constraints of the problem).

Note that, even though the propagation process does not explicitly consider consistency with respect to the resource constraints, it is guaranteed to obtain a

[3] Although the reader should also note that these solutions are in general sub-optimal.

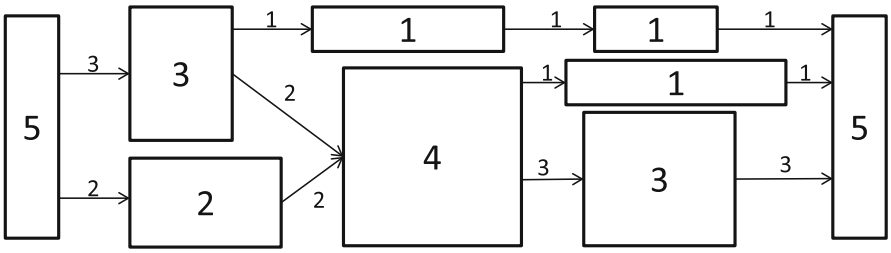

Fig. 6.4 An example of Partial Order Schedule (*POS*)

feasible solution by definition. Therefore a partial order schedule provides a means to find a new solution and ensures to compute it in a fast way.

The common thread underlying a *POS* 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 constraint, but represents a *producer-consumer* relation, allowing each activity to be connected with the 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 completes its execution, it passes its resource unit(s) on to its successors (a similar formulation is used in Chap. 2 of this handbook). Figure 6.4 shows an example of Partial Order Schedule for a single resource with capacity five. In particular, activities are represented as rectangles and edges represent the precedence constraints. The numbers inside the rectangles represent the resource requirements and the labeling numbers on the directed edges represents the flow of resource units supplied to a generic activity i from its predecessors in order to satisfy the imposed resource constraint, independently from the start time values of the activities. For example, the activity which requires four units of resource receives two units of resource from each of its two predecessors and supplies one and three units of resource respectively to its two successors. In general, in a *POS* solution each activity has a set of inputs predecessors which supply the units of resource needed for its execution.

Hence, an activity network $N_S(V_S, E_{POS})$ is in *POS-form* if for each resource k there exists a labeling function $f_k : E_{POS} \longrightarrow [0..R_k]$ representing the flow of resource units among the activities such that for each activity i the following constraint holds:

$$\sum_{j \in Pred(i)} f_k(j,i) = \sum_{j \in Succ(i)} f_k(i,j) = r_{ik} \tag{6.4}$$

where $Pred(i) = \{j \in V | \exists (j,i) \in E_{POS})\}$ and $Succ(i) = \{j \in V | \exists (i,j) \in E_{POS})\}$.[4] Given an input solution S (represented either as a graph or as a set of

[4] $Pred(source) = Succ(sink) = \emptyset$.

start-time values) a polynomial transformation method, named CHAINING, can be defined that creates sets of activity *chains* (Policella et al. 2007). This operation can be accomplished in three steps:

1. all the previously posted leveling constraints are removed from the input partial order;
2. the activities are sorted by increasing activity earliest start times;
3. for each resource and for each activity i (according to the increasing order of start times), one or more predecessors p are chosen, which supplies the units of resource required by i – a precedence constraint $p \prec i$ is posted for each predecessor p. The last step is iterated until all the activities are linked by precedence chains and the constraints in (6.4) are satisfied.

Before concluding we make a further remark about partial order schedules. Roy and Sussman (1964) introduced 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 and transforming each into a conjunctive one. Also in our case of RCPSP/max, solution of all disjunctive constraints is required to achieve a *POS*. In essence, the disjunctive graph representation has been extended to the more general case where multi-capacity resources are defined. In this case "disjunctive" hyper-constraints among activities that use the same resource are introduced, the so-called MCSs. Based on this representation we can note that a partial order schedule is obtained once any disjunctive MCS is solved. In this case, a set of precedence constraints is posted to solve each MCS.

6.8 Extended Optimizing Search

In the previous sections we have presented a basic set of core solving algorithms for generating a solution in the form of an activity network N. In addition, we have also shown as such a network can be always turned into a Partial Order Schedule (*POS*) via a polynomial time calculation. Now in this section we summarize how to use these core algorithms into a set of extended optimizing search procedures targeted on two different objectives: minimize the solution makespan and improve the *robustness* of a solution.

A first procedure is described in (Cesta et al. 2002), where an iterated version of Algorithm 6.2, called the ISES procedure, is proposed. This algorithm is an iterative method, which at each step utilizes a *randomized* version of Algorithm 6.2 to produce different solutions. The key idea underlying the described approach is to heuristically bias random choices in a dynamic fashion, according to how well (or how poor) the available search heuristics (variable and value ordering) discriminate among several alternatives.

A generalization of the previous procedure is the so-called *Iterative Flattening Search* (IFS, Cesta et al. 2000; Oddi et al. 2010a). The concept of iterative flattening search is quite general and provides a framework for designing effective procedures

Algorithm 6.3: IFS(S, *MaxFail*)

```
S_best ← S
counter ← 0
while counter ≤ MaxFail do
    Relax(S)
    S ← PCP(S)
    if C_max(S) < C_max(S_best) then
        S_best ← S
        counter ← 0
    else
        counter ← counter + 1
    end if
end while
return S_best
```

for scheduling optimization, this concept is also known in literature as Large Neighborhood Search (LNS) and was independently developed for solving vehicle routing problems in Shaw (1998). It iteratively applies two steps: (1) Random relaxation of the current solution; (2) An incremental solving step to regain solution feasibility. Algorithm 6.3 introduces the generic IFS procedure. The algorithm alternates relaxation and flattening steps until a better solution is found or a maximal number of non-improving iterations is reached. The procedure takes two parameters as input: (1) an initial solution S; (2) a positive integer *MaxFail*, which specifies the maximum number of consecutive non makespan-improving moves that the algorithm will tolerate before terminating. After initialization, a solution is repeatedly modified within the while loop by applying the RELAX procedure, and a PCP procedure is used as solving step. At each iteration, the RELAX step reintroduces the possibility of resource contention, and the PCP step is called again to restore resource feasibility. If a better makespan solution is found, the new solution is saved in S_{best}. If no improvement is found within *MaxFail* moves, the algorithm terminates and returns the best solution found. It is worth noting that Partial Order Schedules in the context of IFS (or LNS) are an effective way for designing neighborhood structures, examples of neighborhoods for solving scheduling problems with cumulative renewable resources are given in the papers Oddi et al. (2010a), in addition, as shown in Laborie and Godard (2007), the concept can be extended to various types of resources.

The problem of increasing (optimizing) the robustness of generated *POS*s is addressed via an iterative (randomized) chaining procedure in Policella et al. (2009). In particular, the problem is addressed by separating the phase of problem solution, which may pursue a standard optimization criterion (e.g., minimal makespan), from a subsequent phase of solution *robustification* in which a more flexible set of solutions is obtained and compactly represented through a Partial Order Schedule. In particular, the paper focuses on specific heuristic algorithms for synthesis of *POS*s, starting from a pre-existing schedule and different extensions of the technique CHAINING algorithm described in Sect. 6.7.1, which progressively

introduces temporal flexibility into the representation of the solution. In fact, we can observe that given the form of the output solution (an activity network), "classical" objective like the minimization of project makespan, C_{max}, co-exists very naturally with a *POS* solution and the objective of increasing the solution's robustness. In fact, in the best case, the early start time solution which minimizes makespan is executed; but in the event that unexpected events prevent this possibility, the accompanying *POS* provides a feasible, bounded relaxation strategy.

6.9 Conclusions

In this chapter we have summarized a constraint satisfaction problem solving (CSP) framework for solving project scheduling problems. We have advocated a somewhat unconventional Precedence Constraint Posting (PCP) approach, which exploits the expressiveness and computational efficiency of a simple temporal network (STN) to enforce complex temporal constraints and interdependencies between activities, and establishes resource feasibility by iteratively sequencing activities that are found to be competing for the same resources until all potential resource conflicts have been resolved. One important advantage of a PCP approach is that a generated solution consists of a set of feasible schedules, as opposed to a single assignment of activity start times that is the target of many scheduling approaches. In fact, such a flexible solution can be efficiently transformed into a Partial Order Schedule (*POS*), which guarantees resource feasibility over ranges of activity start times that are consistent with posted constraints. *POS*s can be enable quick reaction to unforeseen events during execution via efficient temporal constraint propagation procedures.

We have discussed core technology components for managing complex temporal constraints and for detecting and responding to potential resource conflicts, which are necessary ingredients for configuring a PCP-based scheduling procedure. We have also illustrated their applicability to the resource constrained project scheduling problem with minimum and maximum lag times (RCPSP/max). Due to space considerations, we have not focused heavily on the issue of optimization of project schedules. However, it is important to note that the PCP approach we have described can be directly embedded as a sub-procedure in various forms of extended optimizing search (see Cesta et al. 2002; Oddi et al. 2010a; Policella et al. 2009).

Minimization of project makespan, for example, is an objective that co-exists very naturally with a *POS* solution—In the best case, the early start time solution which minimizes makespan is executed; but in the event that unexpected events prevent this possibility, the accompanying *POS* provides a feasible, bounded relaxation strategy. Overall, PCP provides a flexible and efficient framework for solving complex combinatorial problems like project scheduling.

Acknowledgements Authors would like to thank the anonymous reviewers for detailed comments to previous drafts of this chapter. Stephen Smith was supported in part by the US Air Force

Research Laboratory under contracts FA8650-12-C-6269 and FA8750-12-C-0068, and the CMU Robotics Institute. CNR authors were supported by internal funds for basic research.

References

Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manage Sci 34(3):391–401

Baptiste P, Le Pape C (1995) A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In: IJCAI-95. Morgan Kaufmann, San Francisco, pp 600–606

Baptiste P, Le Pape C, Nuijten W (2001) Constraint-based scheduling. Kluwer, Boston

Bartusch M, Mohring RH, Radermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16(1):201–240

Beck JC, Davenport AJ, Davis ED, Fox MS (1998) The ODO project: towards a unified basis for constraint-directed scheduling. J Sched 1:89–125

Cesta A, Stella C (1997) A time and resource problem for planning architectures. In: ECP-97. Lecture notes in computer science, vol 1348. Springer, New York, pp 117–129

Cesta A, Oddi A, Smith SF (1999) An iterative sampling procedure for resource constrained project scheduling with time windows. In: IJCAI-99. Morgan Kaufmann, San Francisco, pp 1022–1029

Cesta A, Oddi A, Smith SF (2000) Iterative flattening: a scalable method for solving multi-capacity scheduling problems. In: AAAI-00. AAAI Press, Menlo Park, pp 742–747

Cesta A, Oddi A, Smith SF (2002) A constraint-based method for project scheduling with time windows. J Heuristics 8(1):109–136

Davenport AJ, Gefflot C, Beck JC (2001) Slack-based techniques for robust schedules. In: ECP-01. Lecture notes in computer science. Springer, Heidelberg, pp 7–18

Dechter R, Rossi F (2002) Constraint satisfaction. In: Nadel L (ed) Encyclopedia of cognitive science, Nature Publishing Group, London

Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artif Intell 49(1–3):61–95

Drabble B, Tate A (1994) The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In: AIPS-94. AAAI Press, Menlo Park, pp 243–248

Drummond M, Bresina J, Swanson K (1994) Just-in-case scheduling. In: AAAI-94. AAAI Press, Menlo Park, pp 1098–1104

El Sakkout HH, Wallace MG (2000) Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 5(4):359–388

Fox MS (1990) Constraint guided scheduling: a short history of scheduling research at CMU. Comp Ind 14(1–3):79–88

Hentenryck PV, Michel L (2009) Constraint-based local search. MIT Press, Cambridge

Hiatt LM, Zimmerman TL, Smith SF, Simmons R (2009) Strengthening schedules through uncertainty analysis agents. In: IJCAI-09. AAAI Press, Menlo Park

Kumar V (1992) Algorithms for constraint-satisfaction problems: a survey. Artif Intell Mag 13(1):32–44

Laborie P (2003) Algorithms for propagating resource constraints in A.I. planning and scheduling: existing approaches and new results. Artif Intell 143(2):151–188

Laborie P, Ghallab M (1995) Planning with sharable resource constraints. In: IJCAI-95. Morgan Kaufmann, San Francisco, pp 1643–1651

Laborie P, Godard D (2007) Self-adapting large neighborhood search: application to single-mode scheduling problems. In: Proceedings MISTA-07, Paris, pp 276–284

Montanari U (1974) Networks of constraints: fundamental properties and applications to picture processing. Inform Sci 7:95–132

Muscettola N (2002) Computing the envelope for stepwise-constant resource allocations. In: CP-2002. Lecture notes in computer science, vol 2470. Springer, Heidelberg, pp 139–154

Nuijten WPM, Aarts EHL (1996) A computational study of constraint satisfaction for multiple capacitated job shop scheduling. Eur J Oper Res 90(2):269–284
Nuijten W, Le Pape C (1998) Constraint-based job shop scheduling with ILOG-scheduler. J Heuristics 3(4):271–286
Oddi A, Cesta A, Policella N, Smith S (2010a) Iterative flattening search for resource constrained scheduling. J Intell Manuf 21(1):17–30
Oddi A, Rasconi R, Cesta A (2010b) Project scheduling as a disjunctive temporal problem. In: ECAI 2010. IOS Press, Amsterdam, pp 967–968
Policella N (2005) Scheduling with uncertainty: a proactive approach using partial order schedules. Ph.D. dissertation, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome
Policella N, Smith SF, Cesta A, Oddi A (2004) Generating robust schedules through temporal flexibility. In: ICAPS'04. AAAI Press, Menlo Park, pp 209–218
Policella N, Cesta A, Oddi A, Smith S (2007) From precedence constraint posting to partial order schedules: a CSP approach to robust scheduling. AI Commun 20(3):163–180
Policella N, Cesta A, Oddi A, Smith S (2009) Solve-and-robustify. J Sched 12(3):299–314
Rossi F, van Beek P, Walsh T (2006) Handbook of constraint programming. Foundations of artificial intelligence, Elsevier Science, Amsterdam
Roy B, Sussman B (1964) Les problemes d'ordonnancement avec contraintes disjonctives, note DS n. 9 bis. SEMA, Paris
Sadeh NM (1991) Look-ahead techniques for micro-opportunistic job shop scheduling. Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh
Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: CP98. Lecture notes in computer science, vol 1520. Springer, Berlin, pp 417–431
Smith SF (1994) OPIS: a methodology and architecture for reactive scheduling. In: Fox M, Zweben M (eds) Intelligent scheduling, Morgan Kaufmann, San Francisco, pp 29–66
Smith SF, Cheng C (1993) Slack-based heuristics for constraint satisfactions scheduling. In: AAAI-93. AAAI Press, Menlo Park, pp 139–144
Smith TB, Pyle JM (2004) An effective algorithm for project scheduling with arbitrary temporal constraints. In: AAAI'04. AAAI Press, Menlo Park, pp 544–549
Tsang EPK (1993) Foundations of constraint satisfaction. Academic Press, London/San Diego