Handbook of Constraint Programming
Edited by F. Rossi, P. van Beek and T. Walsh

Chapter 22

Constraint-Based Scheduling and Planning

Philippe Baptiste, Philippe Laborie, Claude Le Pape, Wim Nuijten

Solving a scheduling problem generally consists in allocating scarce resources to a given set of activities over time. Planning can be seen as a generalization of scheduling where the set of activities to be scheduled is not known in advance. The additional complexity of planning thus lies in the fact one also has to decide on the set of activities that will be scheduled. Constraint-Based Scheduling is the discipline that studies how to solve scheduling problems by using Constraint Programming (CP). Constraint-Based Planning in turn is the discipline that studies how to solve planning problems by CP. As the use of CP in scheduling is more mature, we first turn to Constraint-Based Scheduling after which we will come back to Constraint-Based Planning.

Constraint-Based Scheduling has over the years grown into one of the most successful application areas of CP. One of the key factors of this success lies in the fact that a combination was found of the best of two fields of research that pay attention to scheduling, namely *Operations Research* (OR) and *Artificial Intelligence* (AI). Traditionally, a lot of the attention in OR has been paid to rather "pure" scheduling problems that are based on relatively simple mathematical models. For solving the problem at hand, the combinatorial structure of the problem is heavily exploited, leading to improved performance characteristics. We could say that an OR approach often aims at achieving a high level of *efficiency* in its algorithms. However, when modeling a practical scheduling problem using these classical models, one is often forced to discard degrees of freedom and side constraints that exist in the practical scheduling situation. Discarding degrees of freedom may result in the elimination of interesting solutions, regardless of the solution method used. Discarding side constraints gives a simplified problem and solving this simplified problem may result in impractical solutions for the original problem.

In contrast, AI research tends to investigate more general scheduling models and tries to solve the problems by using general problem-solving paradigms. We could say an AI approach tends to focus more on the *generality of application* of its algorithms. This, however, implies that AI algorithms may perform poorly on specific cases, compared to OR algorithms.

So, on one hand we have OR which offers us *efficient* algorithms to solve problems that in comparison have a more limited application area. On the other hand we have AI that offers us algorithms that are more *generally applicable*, but that might suffer from somewhat poor performance in the specific cases an efficient OR algorithm exists. An important way to combine the two was found by incorporating OR algorithms inside *global* constraints. Such algorithms are able to take into account a set of constraints from a global point of view in an efficient way. The typical scheduling example of a global constraint is the constraint that propagates on the combination of all activities requiring capacity from a shared resource. The basics of many of the algorithms inside global constraints, certainly in the early stages of the field, can be found in OR. By applying the locality principle [70], such specialized algorithms can work side by side with general propagation algorithms that take care of the rest of the constraints. In this way one can preserve the general modeling and problem-solving paradigm of CP while the integration of efficient propagation algorithms improves the overall performance of the approach. Stated in another way, efficient OR algorithms integrated in a CP approach allow the user to benefit from the efficiency of OR techniques in a flexible framework. Translated to the area of Constraint-Based Scheduling two strengths emerge: i) natural and flexible modeling of scheduling problems as Constraint Satisfaction Problems (CSPs) [72] and ii) powerful propagation of temporal and resource constraints. All this said, we want to remark that over the years the distinction between AI and OR is often becoming less and less clear and is also deemed less and less important.

Let us now step from Constraint-Based Scheduling to Constraint-Based Planning. There are basically two approaches for applying CP to planning:

- The first approach (see Section 22.2.2) dates back to the first attempts to build non-linear plans in the 70s [62, 17, 53] and consists in refining a partial plan made of a temporal network of activities. Similarly to Constraint-Based Scheduling, constraint propagation can be used in this temporal network to propagate temporal, state, and resource constraints. These approaches are flexible enough to handle *complex and realistic* planning problems [66]. Although in these approaches the idea of constraint propagation has been present since the beginning, it is only in recent years that efficient global propagation algorithms, partially inspired from the ones available in Constraint-Based Scheduling have been designed [75].

- The second approach (see Section 22.2.1) consists in compiling the planning problem into a CSP and use CSP or SAT solvers as a blackbox to solve the problem [6, 73, 23, 51]. These approaches work on a simplification of the real planning problem expressed in a STRIPS-like formalism and tend to focus on *efficiency* rather than on generality of application. Several ideas stemming from this approach can be used to provide efficient global constraint propagation algorithms and heuristics to guide the search [75].

Although the use of CP in planning is, due to the problem complexity, less mature than its use in scheduling, Constraint-Based Planning thus follows the same pattern as

Constraint-Based Scheduling where CP is used as a framework for integrating efficient special purpose algorithms into a flexible and expressive paradigm.

Besides the strengths mentioned above, we want to mention two more reasons that contributed to the success of Constraint-Based Scheduling: a natural fit of expressing scheduling specific heuristics with CP tree search, and a proven good potential of combining the CP approach with solution techniques as Local Search, Large Neighborhood Search, and Mixed Integer Programming. The first allows to both exploit all the work done on scheduling heuristics in the past as to write new scheduling heuristics with easy to understand scheduling semantics. The second allows to get improved performance in the cases where that is needed. The same strengths should also benefit Constraint-Based Planning.

The remainder of this chapter is organized as follows. Section 22.1 presents CP models for scheduling together with descriptions of propagation techniques for constraints used in these models. Section 22.2 does the same for planning problems. As over the years dedicated global constraint propagation techniques for resource constraints have received a lot of attention, they are discussed in more detail in Section 22.3. In Section 22.4 constraint propagation techniques on optimization criteria are discussed, after which Section 22.5 pays attention to the search procedures that are used in Constraint-Based Planning and Scheduling to solve resource constraints. Finally in Section 22.6 conclusions are presented together with identification of potential future research directions.

22.1 Constraint Programming Models for Scheduling

In this section we give an overview of the kind of scheduling problems that are studied by the Constraint-Based Scheduling community. We present the different components together with a description of how these components can be modeled as a part of a CSP.

Throughout this chapter we use the following notation. Let $\{A_1, \ldots, A_n\}$ be a set of n activities and $\{R_1, \ldots, R_m\}$ a set of m resources. Let's for now consider a basic scheduling problem where each of the activities has a processing time and requires a certain capacity from one or several resources. The resources have a given capacity that can not be exceeded at any point in time. There may furthermore be a set of temporal constraints between activities and an objective function. The problem to be solved is to decide when to execute each activity to optimize the objective function, while respecting both temporal and resource constraints. Later on in this section several extensions are discussed, but for now this basic scheduling problem suffices for the discussion.

22.1.1 Activities

When looking at the type of activities in a problem, we distinguish *non-preemptive scheduling*, *preemptive scheduling*, and *elastic scheduling*. In non-preemptive scheduling, activities cannot be interrupted. Each activity must execute without interruption from its start time to its end time. In preemptive scheduling, activities can be interrupted at any time, *e.g.*, to let some other activities execute. In elastic scheduling the amount of resource assigned to an activity A_i can, at any time t, assume any value between 0 and the resource capacity, provided that the sum over time of the assigned capacity equals a given value called *energy*. The equivalent notion of energy in the case of a non-preemptive activity is the product of its processing time and the capacity required.

A non-preemptive scheduling problem can be efficiently encoded as a CSP in the following way. For each activity three variables are introduced, $\mathit{start}(A_i)$, $\mathit{end}(A_i)$, and $\mathit{proc}(A_i)$. They represent the start time, the end time, and the processing time of A_i, respectively. We are not aware of CP approaches not using this encoding.

With r_i the release date and d_i the deadline of activity A_i as defined in the initial data of the scheduling problem, $[r_i, d_i]$ is the time window in which A_i has to execute. Based on that the initial domains of $\mathit{start}(A_i)$ and $\mathit{end}(A_i)$ are $[r_i, \mathit{lst}_i]$ and $[\mathit{eet}_i, d_i]$, respectively. Here lst_i and eet_i stand for the latest start time and the earliest end time of A_i. For convenience, we also use this notation to denote the current domains of $\mathit{start}(A_i)$ and $\mathit{end}(A_i)$, *i.e.*, the domains when we are in the process of propagating constraints. Of course in that case instead of the initial release date and deadline, r_i and d_i denote the current earliest start time and latest end time

The processing time of the activity is defined as the difference between the end time and the start time of the activity: $\mathit{proc}(A_i) = \mathit{end}(A_i) - \mathit{start}(A_i)$. p_i denotes the smallest value in the domain of $\mathit{proc}(A_i)$. All data related to an activity are summarized in Figure 22.1. Light gray is used to depict the time window $[r_i, d_i]$ of an activity and dark gray is used to represent the processing time of the activity.

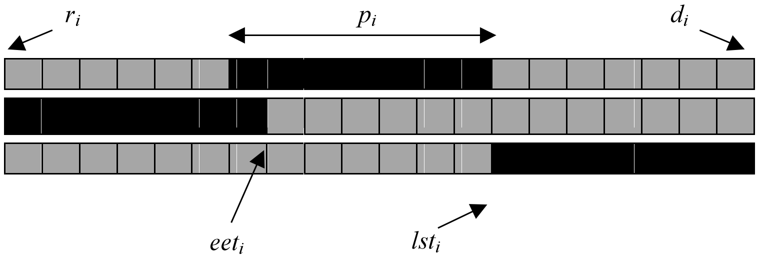


Figure 22.1: Data related to an activity

Preemptive scheduling problems are more difficult to represent since a schedule is more complex than a simple set of start and end times of activities. We discuss two possibilities. One can either associate a set variable (*i.e.*, a variable the value of which will be a set) $\mathit{set}(A_i)$ with each activity A_i, or alternatively define a 0-1 variable $X(A_i, t)$ for each activity A_i and time t. $\mathit{set}(A_i)$ represents the set of times at which A_i executes, while $X(A_i, t)$ takes value 1 if and only if A_i executes at time t. The processing time $\mathit{proc}(A_i)$ of A_i is defined as the number of time points t at which A_i executes, *i.e.*, as $|\mathit{set}(A_i)|$. In practice, the $X(A_i, t)$ variables are not represented explicitly as the value of $X(A_i, t)$ is 1 if and only if t belongs to $\mathit{set}(A_i)$.

Assuming time is discretized, $\mathit{start}(A_i)$ and $\mathit{end}(A_i)$ can be defined by $\mathit{start}(A_i) = \min_{t \in \mathit{set}(A_i)} t$ and $\mathit{end}(A_i) = \max_{t \in \mathit{set}(A_i)} t + 1$. Notice that in the non-preemptive case, $\mathit{set}(A_i) = [\mathit{start}(A_i), \mathit{end}(A_i))$, with the interval $[\mathit{start}(A_i), \mathit{end}(A_i))$ closed on the left and open on the right so that $|\mathit{set}(A_i)| = \mathit{end}(A_i) - \mathit{start}(A_i) = \mathit{proc}(A_i)$.

These constraints are easily propagated by maintaining a lower bound and an upper bound for the set variable $\mathit{set}(A_i)$. The lower bound $\mathit{lb}(\mathit{set}(A_i))$ is a series of disjoint intervals ILB_i^u such that each ILB_i^u is constrained to be included in *set*(A_i). The upper bound *ub*(*set*(A_i)) is a series of disjoint intervals IUB_i^v such that *set*(A_i) is constrained to be included in the union of the IUB_i^v. If the size of the lower bound (*i.e.*, the sum of the sizes of the ILB_i^u) becomes larger than the upper bound of *proc*(A_i) or if the size of the upper bound (*i.e.*, the sum of the sizes of the IUB_i^v) becomes smaller than the lower bound of *proc*(A_i), a contradiction is detected. If the size of the lower bound (or of the upper bound) becomes equal to the upper bound (respectively, lower bound) of *proc*(A_i), *set*(A_i) receives the lower bound (respectively, the upper bound) as its final value. Minimal and maximal values of *start*(A_i) and *end*(A_i), *i.e.*, earliest and latest start and end times, are also maintained. Each of the following rules, considered independently one from another, is used to update the bounds of *set*(A_i), *start*(A_i) and *end*(A_i). Let t be any point in time, then

$$\begin{aligned}
&t < r_i \Rightarrow t \notin \mathit{set}(A_i)\\
&t \in \mathit{lb}(\mathit{set}(A_i)) \Rightarrow \mathit{start}(A_i) \le t\\
&d_i \le t \Rightarrow t \notin \mathit{set}(A_i)\\
&t \in \mathit{lb}(\mathit{set}(A_i)) \Rightarrow t < \mathit{end}(A_i)\\
&[\forall_{u<t}\ u \notin \mathit{ub}(\mathit{set}(A_i))] \Rightarrow t \le \mathit{start}(A_i)\\
&[\forall_{u\ge t}\ u \notin \mathit{ub}(\mathit{set}(A_i))] \Rightarrow \mathit{end}(A_i) \le t\\
&\mathit{start}(A_i) \le \max\{u \mid \exists_{S\subseteq \mathit{ub}(\mathit{set}(A_i))}\ |S| = p_i \wedge \min(S) = u\}\\
&\mathit{end}(A_i) \ge \min\{u \mid \exists_{S\subseteq \mathit{ub}(\mathit{set}(A_i))}\ |S| = p_i \wedge \max(S) = u-1\}
\end{aligned}$$

Needless to say, whenever any of these rules leads to a situation where the lower bound of a variable is larger than its upper bound, a contradiction is detected.

In the following, we may occasionally use the notations $X(A_i, t)$ and *set*(A_i) for an activity A_i that cannot be interrupted. In such a case, the following rules are also applied:

$$\begin{aligned}
&X(A_i,t) = 0 \wedge t < \mathit{eet}_i \Rightarrow \mathit{start}(A_i) > t\\
&X(A_i,t) = 0 \wedge \mathit{lst}_i \le t \Rightarrow \mathit{end}(A_i) \le t
\end{aligned}$$

Elastic activities are discussed in the following section.

22.1.2 Resource Constraints

When looking at the type of resources found in a problem, we distinguish *disjunctive scheduling* and *cumulative scheduling*. In a disjunctive scheduling problem, all resources are of capacity 1 (such resources are often called *machines*) and thus can execute at most one activity at a time. In a cumulative scheduling problem, resources exist that can execute several activities in parallel, of course provided that the resource capacity is not exceeded.

Resource constraints represent the fact that activities require some amount of resource throughout their execution. Given an activity A_i and a resource R whose capacity is *cap*(R), generally a variable *cap*(A_i, R) is introduced that represents the amount of resource R required by activity A_i. Where no confusion is possible we will often omit "R" and use *cap*(A_i) to denote *cap*(A_i, R). For fully elastic activities, the *cap*(A_i, R) variable is not meaningful and a variable $E(A_i, R)$ that represents the *energy* required by the activity on the resource R is introduced. Note that for non-elastic activities, we have

$E(A_i, R) = \mathit{cap}(A_i, R)\mathit{proc}(A_i)$. To represent a schedule, a set of variables $E(A_i, t, R)$ is required that denote the number of units of the resource R used by activity A_i at time t. In all cases, we have the constraint stating that enough resource capacity must be allocated to activities to cover the energy requirement:

$$E(A_i, R) = \sum_t E(A_i, t, R)$$

If A_i is not an elastic activity, there are some strong relations between $E(A_i, t, R)$ and $X(A_i, t)$:

$$E(A_i, t, R) = X(A_i, t)\mathit{cap}(A_i, R)$$

For elastic activities, we have a weaker relation between the variables:

$$[E(A_i, t, R) > 0] \Leftrightarrow [X(A_i, t) > 0]$$

Generally speaking, the resource constraint can be written as follows. For each point in time t

$$\sum_{i=1}^{n} E(A_i, t, R) \leq \mathit{cap}(R) \tag{22.1}$$

Depending on the scheduling situation, (22.1) can be rewritten. In the non-preemptive case, (22.1) leads for all times t to

$$\sum_{A_i \mid \mathit{start}(A_i) \leq t < \mathit{end}(A_i)} \mathit{cap}(A_i, R) \leq \mathit{cap}(R)$$

In the preemptive case, (22.1) leads for all times t to

$$\sum_{A_i \mid \mathit{start}(A_i) \leq t < \mathit{end}(A_i)} X(A_i, t)\mathit{cap}(A_i, R) \leq \mathit{cap}(R)$$

22.1.3 Temporal Constraints

Temporal relations between activities can be expressed by linear constraints between start and end variables of activities. For instance, a standard precedence constraint between two activities A_1 and A_2 stating that A_2 is to be started after A_1 is ended can be modeled by the linear constraint $\mathit{end}(A_1) \leq \mathit{start}(A_2)$. In general, with both x and y a start or end variable and d an integer, temporal relations can be expressed by constraints of the type $x - y \leq d$.

When the temporal constraint network is sparse, as it is usually the case in scheduling, such constraints can be easily propagated using a standard arc-B-consistency algorithm [49]. In addition, a variant of Ford's algorithm (see for instance [37]) proposed by Cesta and Oddi [15] can be used to detect any inconsistency between such constraints, in time polynomial in the number of constraints and independent of the domain sizes.

When the temporal network is dense or when it is useful to compute and maintain the minimal and maximal delay between any pair of time points in the schedule, path consistency can be enforced on the network [21] for example by applying Floyd-Warshall's All-Pairs-Shortest-Path algorithm [27].

22.1.4 Extensions of the Basic Model

Although the model presented until now covers quite a number of scheduling problems, in this section we pay attention to extensions that are frequently found in industrial applications.

Alternative resources

In some scheduling situations an activity A_i can be scheduled on any one resource from a set S of resources. We say that S is the set of *alternative resources* for A_i. A common way to model this is to for each activity A_i introduce a variable $\mathit{altern}(A_i)$ representing the chosen resource among the resource alternatives. To simplify notation, we assume that resources are numbered from 1 to m and that $\mathit{altern}(A_i)$ denotes the variable whose value represents the index of the resource on which A_i is executed. We remark that quite commonly the processing time of the activity depends on the resource on which the given activity is executed, *i.e.*, the resources are unrelated. The same goes for the cost of executing the activity, *i.e.*, different alternatives can have different costs. Another commonly found type of constraints reasons on interdependencies of resource allocations, *e.g.*, a constraint like "if A_1 is scheduled on resource R_1 then A_3 has to be scheduled on resource R_2". These constraints are used to model things like alternative production lines.

Alternative resource constraints are propagated as if the activity A_i were split into $|\mathit{domain}(\mathit{altern}(A_i))|$ fictive activities A_i^u where each activity A_i^u requires resource R_u [47]. Following this notation r_i^u denotes the earliest start time of A_i^u, *etc*. The alternative resource constraint maintains the constructive disjunction between the alternative activities A_i^u for $u \in \mathit{domain}(\mathit{altern}(A_i))$, *i.e.*, it ensures that:

$$\begin{array}{rcl}
r_i & = & \min\{r_i^u \mid u \in \mathit{domain}(\mathit{altern}(A_i))\} \\
\mathit{lst}_i & = & \max\{\mathit{lst}_i^u \mid u \in \mathit{domain}(\mathit{altern}(A_i))\} \\
\mathit{eet}_i & = & \min\{\mathit{eet}_i^u \mid u \in \mathit{domain}(\mathit{altern}(A_i))\} \\
d_i & = & \max\{d_i^u \mid u \in \mathit{domain}(\mathit{altern}(A_i))\} \\
\mathit{lb}(\mathit{proc}(A_i)) & = & \min\{\mathit{lb}(\mathit{proc}(A_i^u)) \mid u \in \mathit{domain}(\mathit{altern}(A_i))\} \\
\mathit{ub}(\mathit{proc}(A_i)) & = & \max\{\mathit{ub}(\mathit{proc}(A_i^u)) \mid u \in \mathit{domain}(\mathit{altern}(A_i))\}
\end{array}$$

Constraint propagation will deduce new bounds for alternative activities A_i^u on the alternative resource R_u. Whenever the bounds of an activity A_i^u turn out to be incoherent, the resource R_u is simply removed from the set of possible alternative resources for activity A_i, *i.e.*, $\mathit{domain}(\mathit{altern}(A_i))$ becomes $\mathit{domain}(\mathit{altern}(A_i)) - \{u\}$.

In some approaches the fictive activities A_i^u are actually generated together with a way to express that only one of them per original activity A_i will really require one of the alternative resources. In this context the generated activities A_i^u are often referred to as *optional* activities. See below for a discussion on optional activities.

Setup times and setup costs

Setup times and setup costs are of great importance in industrial applications. They are found abundantly and the correct treatment of them is often crucial, both because they are a mandatory component to express the problem in the required detail as it is needed to find good solutions with respect to them as they represent a substantial part of the real-life cost. The setup time (also transition time) *setup*(A_1, A_2) between two activities A_1 and A_2 is defined as the amount of time that must elapse between the end of A_1 and the start of A_2, when A_1 immediately precedes A_2 on a given resource. A setup cost *setupCost*(A_1, A_2) can also be associated to the transition between A_1 and A_2. The objective of the scheduling problem can be to find a schedule that minimizes the sum of the setup costs.

In a vast majority of problems activities subjected to setups are to be scheduled on the same machine (the semantics of setups is much more complex on resources of capacity greater than 1). Setup considerations can be combined with alternative resources. In such a case, two parameters are associated to each tuple (A_i, A_j, R_u): the setup time *setup*(A_i, A_j, R_u) and the setup cost *setupCost*(A_i, A_j, R_u) between activities A_i and A_j if A_i and A_j are scheduled sequentially on the same machine R_u. The attached constraint is that *start*$(A_j^u) \geq$ *end*$(A_i^u) +$ *setup*(A_i, A_j, R_u). There may furthermore exist a setup time *setup*$(-, A_i, R_u)$ (with corresponding cost *setupCost*$(-, A_i, R_u)$) that has to elapse before the start of A_i when A_i is the first activity on R_u and, similarly, a teardown time *setup*$(A_i, -, R_u)$ (with corresponding cost *setupCost*$(A_i, -, R_u)$) that has to elapse after the end of A_i when A_i is the last activity on R_u. Section 22.4.2 pays attention to constraint propagation methods for setup times and setup costs constraints.

Breakable activities and calendars

In [58] a problem is described that has a lot of properties frequently found in industrial applications. One such property is the fact resources can be governed by a *calendar* under which activities scheduled on the resource are executed. Such a calendar thus defines the execution conditions for activities and consists of a list of breaks and a *productivity profile*. An activity scheduled on the resource at hand can be interrupted by breaks smaller than the maximal break duration mBD. An activity is thus *breakable* when $mBD > 0$ and *not breakable* when $mBD = 0$. The productivity profile defines the efficiency of the activity execution. If an activity is scheduled in a time interval with productivity $p\%$, $p\%$ of a processing time unit is executed per time unit. A productivity below 100% thus means that per time unit less than one processing time unit is executed which in turn implies that the duration of an activity will exceed its processing time. For productivities exceeding 100% obviously the inverse holds. The processing time of an activity is therefore equal to the integral, from start to end, of the productivity. We refer to [65] for a more extensive discussion on breakable activities and productivity profiles.

A CP model for this consists in introducing a *duration variable* per activity and redefining the meaning of the processing time variable. The duration variable *dur*(A_i) of an activity is then defined as the difference between the end time and the start time of the activity: *dur*$(A_i) =$ *end*$(A_i) -$ *start*(A_i). The processing time variable is defined as the time it will take to execute the activity when it is not interrupted by breaks and all along the execution the productivity is exactly 100%. All four activity variables *start*, *end*, *dur*, and *proc* are governed by the break constraint and the productivity profile constraint of the resource on which the activity is executed. For sake of clarity we do not use this meaning for the processing time variable in the rest of this chapter, *i.e.*, the processing time is defined as the difference between the end time and the start time of an activity.

Optional activities/leaving activities unperformed

Whether coming from alternative resources (see above) or directly present in the model, in many scheduling problems activities exist for which it is not yet decided whether they will be executed on a resource or not. Such activities are often referred to as *optional* activities.

Modeling an optional activity is often obtained by allowing the processing time variable to take on the value 0. Care is to be taken then that non-standard precedence constraints are not still active. If for instance an optional activity A_1 is part of a chain of activities and a precedence constraint of the type $\mathit{end}(A_1) + d \leq \mathit{start}(A_2)$ is defined, setting the processing time variable to become 0 and keeping the precedence constraint active will still induce a possibly unwanted delay d between the predecessors of A_1 and A_2. Another way to model optional activities is to introduce a variable per activity expressing whether the activity really exists or not. This is obviously a more direct way of modeling but requires the adaptation of propagation algorithms to deal with this additional variable and concept [78].

In industrial scheduling problems one often finds the possibility of subcontracting an activity A_i, this against a certain incurred cost $cost_i$. This means that the activity does not use resource capacity but does take time. A way to model this is to allow the capacity variable $\mathit{cap}(A_i)$ to take on the value 0 and to introduce a variable $cost$ representing the cost together with the constraint $\mathit{cap}(A_i) = 0 \Rightarrow cost = cost_i$. Another way to model this is to introduce an alternative resource corresponding to the subcontracting alternative. This can be interesting if subcontracting is subject to a different calendar than in-house production, which of course is not uncommon. The cost of the alternative would correspond to $cost_i$. On the "subcontracting" resource one would in this case not enforce the capacity constraint, even though this model can easily be extended to enforce the capacity constraint to model restricted subcontracting possibilities.

In [58] a model is described where one can decide that an activity will be left *unperformed*, meaning that the activity will not require capacity, but will obey potential temporal constraints, etc., and will also obey the calendar of the chosen resource. This corresponds to the situation where one wants to include the possibility to temporarily increase the production capacity in its own production facility, this of course against a certain cost. Other useful constraints included are *performance compatibility constraint* between two activities expressing that either they are both performed or they are both unperformed.

State resources

A state resource represents a resource of infinite capacity, the state of which can vary over time. Each activity may, throughout its execution, require a state resource to be in a given state (or in any of a given set of states). Consequently, two activities may not overlap if they require incompatible states of a state resource during their execution. Adaptations of the Timetable Constraint (see Section 22.3.2) and Disjunctive Constraint (see Section 22.3.1) are used as basic propagation algorithms on those resources.

Reservoirs

A reservoir resource is a multi-capacity resource that can be consumed and/or produced by activities. A reservoir has an integer maximal capacity and may have an initial level. As an example of a reservoir you can think of a fuel tank. Note that a cumulative resource can be seen as a special case of a reservoir that is consumed at the beginning of the activity and produced in the same quantity at the end of the activity when the activity releases the resource.

The Timetable Constraint presented in Section 22.3.2 can be generalized to the case of reservoirs and is classically used as the basic propagation algorithm on those resources. However, we will see in Section 22.3.3 that other techniques are available to provide additional propagation.

22.1.5 Objective Function

Finally, *decision* problems and *optimization* problems are distinguished. In decision problems, one has only to determine whether a schedule exists that meets all constraints. In optimization problems, an objective function has to be optimized. In this chapter we concentrate on problems where there is one objective function, *i.e.*, we do not consider cases where multiple objective functions are defined.

The commonly used way of modeling an objective function is simply by introducing a variable *criterion* that is constrained to be equal to the value of the objective function. Although the minimization of the makespan, *i.e.*, the end time of the schedule, is commonly used, other criteria are of great practical interest *e.g.*, the sum of setup times or costs, the number of late activities, the maximal or average tardiness or earliness, storage costs, alternative costs, the peak or average resource utilization, etc. We will come back to these criteria later.

Many of the classical scheduling criteria take into account a due date δ_i that one would like to meet for each activity. In contrast to a deadline d_i which is mandatory, a due date δ_i can be seen as a preference. In the following, C_i denotes the completion time of activity A_i. Lateness L_i of A_i is defined as the difference between the completion time and the due date of A_i, *i.e.*, $L_i = C_i - \delta_i$. The tardiness T_i of A_i is defined as $\max(0, L_i)$, while earliness of A_i is defined as $\max(0, -L_i)$. The notation U_i is used to denote a unit penalty per late job, *i.e.*, U_i equals 0 when $C_i \leq \delta_i$ and equals 1 otherwise. See also Figure 22.2.

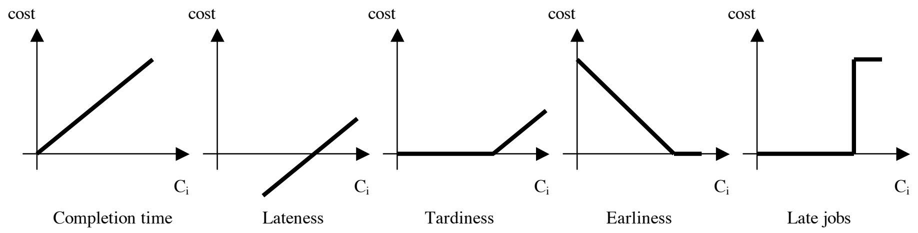


Figure 22.2: Some scheduling related objective functions

The commonly studied criteria F are either formulated as a sum or as a maximum. A weight per activity w_i may be used to give more importance to some activities. We mention the following well-known optimization criteria:

- Makespan: $F = C_{\max} = \max C_i$

- Total weighted flow (or completion) time: $F = \sum w_i C_i$
- Maximum tardiness: $F = T_{\max} = \max T_i$
- Total weighted tardiness: $F = \sum w_i T_i$
- Total weighted number of late jobs: $F = \sum w_i U_i$

For these simple cases which are the most often studied in the literature, the objective function is thus a function of the end variables of the activities.

$$\textit{criterion} = F(\textit{end}(A_1), \ldots, \textit{end}(A_n))$$

In that case, the objective constraint is a simple arithmetic expression on which arc-B-consistency can be easily achieved.

Considering the objective constraint and the resource constraints independently is not a problem when F is a maximum such as $C_{\max}$ or $T_{\max}$. Indeed, the upper bound on *criterion* is directly propagated on the completion time of each activity, *i.e.*, latest end times are tightened efficiently. The situation is much more complex for sum functions such as $\sum w_i C_i$, $\sum w_i T_i$, or $\sum w_i U_i$. For these functions, efficient constraint propagation techniques must take into account the resource constraints and the objective constraint simultaneously. We pay attention to propagation for sum functions in Section 22.4. There we will also pay attention to objective functions that are not a function of the end variables of the activities like sum of setup times and sum of setup costs.

Once all constraints of the problem are added, the most commonly used technique to look for an optimal solution is to solve successive decision variants of the problem. Several strategies can be considered to minimize the value of *criterion*. One way is to iterate on the possible values, either from the lower bound of its domain up to the upper bound until one solution is found, or from the upper bound down to the lower bound determining each time whether there still is a solution. Another way is to use a dichotomizing algorithm, where one starts by computing an initial upper bound $\textit{ub}(\textit{criterion})$ and an initial lower bound $\textit{lb}(\textit{criterion})$ for *criterion*. Then

1. Set $D = \left\lfloor \dfrac{\textit{lb}(\textit{criterion}) + \textit{ub}(\textit{criterion})}{2} \right\rfloor$
2. Constrain *criterion* to be at most D. Then solve the resulting CSP, *i.e.*, determine a solution with $\textit{criterion} \leq D$ or prove that no such solution exists. If a solution is found, set $\textit{ub}(\textit{criterion})$ to the value of *criterion* in the solution; otherwise, set $\textit{lb}(\textit{criterion})$ to $D + 1$.
3. Iterate steps 1 and 2 until $\textit{ub}(\textit{criterion}) = \textit{lb}(\textit{criterion})$.

22.2 Constraint Programming Models for Planning

While the previous section presented CP models for scheduling, in this section we pay attention to CP models for planning as studied by the Constraint-Based Planning community. In this section and in general throughout this chapter we assume that the readers are familiar with basic planning techniques and terminology. For more information on such planning techniques and terminology, we refer to [36].

A general formulation of the planning problem defines three inputs in some formal language:

1. A description of the initial state and expected changes of the world,
2. A description of the agent's goal (*i.e.*, what behavior is desired), and
3. A description of the possible actions that can be performed. This last description is often called a domain theory.

The planner's output is a feasible sequence of actions referred to as a *plan* which, when executed in any world satisfying the initial state and undergoing the expected changes, will achieve the agent's goal.

22.2.1 CSPs for Planning-Graph Techniques

The classical STRIPS [26] representation describes the initial state of world with a complete set of ground propositions. The representation is restricted to goals of attainment, and these goals are defined as a conjunction of propositions; all world states satisfying the goal formula are considered equally good. A domain theory completes a planning problem. In the STRIPS representation, each operator is described with a conjunctive precondition and conjunctive effect that define a transition function from states to states. An action is a fully grounded operator. An action can be executed in any state s satisfying the precondition formula. The effect of executing an action in a state s is described by eliminating from s each proposition of the action delete-list and adding to s each proposition from the action add-list. The add-list and delete-list of the action are called the effect of the action. This defines the so called classical planning problem.

For instance in a blocks world domain, the propositions for describing the state of the world are shown in Table 22.1. The operators of this planning domain move a clear block onto another clear block or the table as shown in Table 22.2.

$clear(X)$:	**There is no block on block X.**
$onT(X)$:	**Block X is on the table.**
$on(X,Y)$:	**Block X is on block Y.**

Table 22.1: Propositions for the blocks world domain

In this section, we will consider the planning problem with initial state: $[on(C,A) \land onT(A) \land onT(B)]$ and goal state: $[on(A,B) \land on(B,C)]$ depicted in Figure 22.3.

In recent years, researchers have investigated the reformulation of planning problems as constraint satisfaction problems (CSPs) in an attempt to use powerful algorithms for constraint satisfaction to find plans more efficiently [73, 23, 51]. In these approaches, each CSP typically represents the problem of finding a plan with a fixed number of steps. A solution to the CSP can be mapped back to a plan; if no solution exists, the number of steps permitted in the plan is increased and a new CSP is generated.

Graphplan [6] works on STRIPS domains by creating a planning graph which represents the set of propositions which can be achieved after a number of steps along with mutual exclusion (mutex) relationships between propositions and actions. Mutually exclusive actions are actions that cannot be executed in the same step. Two actions in the same

BB(X,Y,Z)	**Move X from atop Y to atop Z**
preconditions:	$clear(X) \wedge clear(Z) \wedge on(X,Y)$
add-list:	$clear(Y) \wedge on(X,Z)$
delete-list:	$clear(Z) \wedge on(X,Y)$
TB(X,Y)	**Move X from the table to atop Y**
preconditions:	$clear(X) \wedge clear(Y) \wedge onT(X)$
add-list:	$on(X,Y)$
delete-list:	$clear(Y) \wedge onT(X)$
BT(X,Y)	**Move X from atop Y to the table**
preconditions:	$clear(X) \wedge on(X,Y)$
add-list:	$onT(X) \wedge clear(Y)$
delete-list:	$on(X,Y)$

Table 22.2: Operators for the blocks world domain

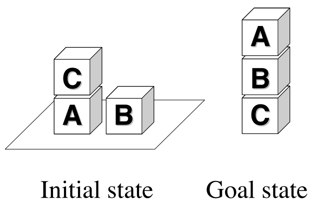


Figure 22.3: A planning problem

step are mutex if either of the actions deletes a precondition or add-effect of the other. Two propositions p and q in the same step are mutex if all possible actions for establishing proposition p are exclusive with all possible actions for establishing proposition q. If two actions a and b have some mutex precondition then they are mutex. Note that the persistence of a proposition between two steps is considered as a particular type of action called a *persistence* action.

This planning graph is then searched for a plan which achieves the goals from the initial state and that is mutex-free in each step. The planning graph corresponding to our blocks world example with 3 developed steps is shown in Figure 22.4. Note that this planning graph is not complete, some mutex are missing in all steps and some actions are missing in step 2.

While the original algorithm performed backward search, the plan graph can also be transformed into a CSP which can be solved by any CSP solver.

In [23], variables of this CSP are propositions distinguished by step. Values are possible actions for establishing those propositions with a special dummy value ($\perp$) stating that the proposition is inactive. Constraints say that preconditions of actions that are used to establish active propositions cannot be inactive and mutex action/proposition pairs must be satisfied. For instance, a constraint $on\_A\_B\_G = TB\_A\_B \Rightarrow (onT\_A_3 \neq \perp \wedge clear\_A_3 \neq \perp \wedge clear\_B_3 \neq \perp)$ means that if proposition on_A_B in the goal state is to be established by action TB_A_B, then, it means that propositions onT_A, $clear\_A$ and $clear\_B$ must hold in step 3, that is, they must have been established by some action. A fragment of the CSP corresponding to the planning graph of Figure 22.4 is shown in Table 22.3.

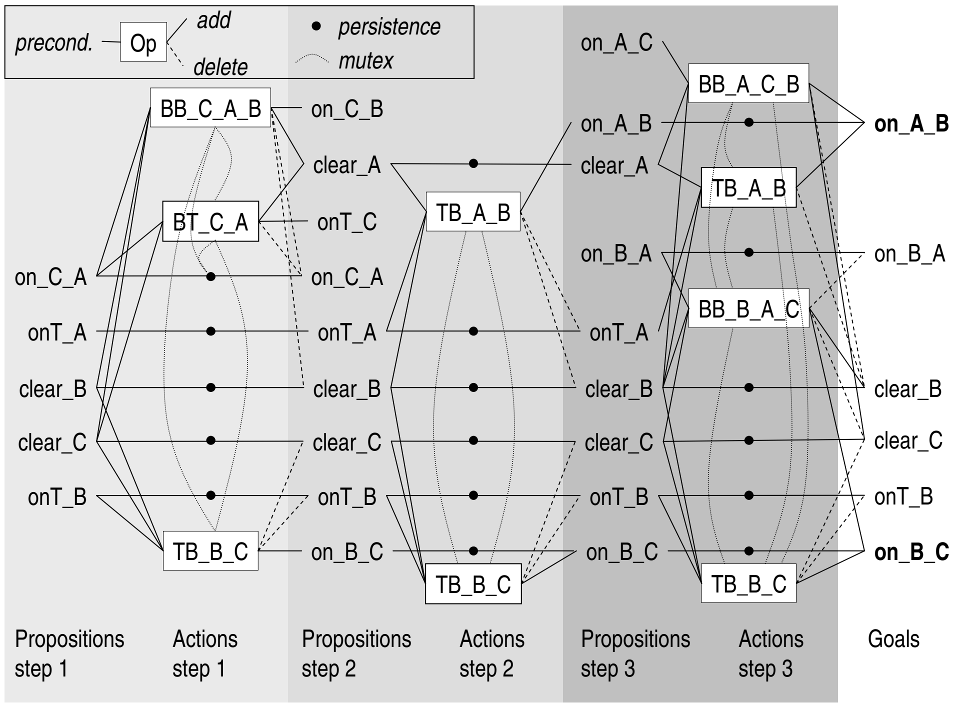


Figure 22.4: Planning graph with 3 steps

Variables:	$on\_C\_A_1, onT\_A_1, clear\_B_1, clear\_C_1, onT\_B_1,$ $on\_C\_B_2, clear\_A_2, onT\_C_2, on\_C\_A_2, onT\_A_2, clear\_B_2, clear\_C_2, onT\_B_2, on\_B\_C_2,$ $on\_A\_C_3, on\_A\_B_3, clear\_A_3, on\_B\_A_3, onT\_A_3, clear\_B_3, clear\_C_3, onT\_B_3, on\_B\_C_3,$ $on\_A\_B_G, on\_B\_C_G$,...
Domains:	$on\_C\_A_1 : \{Init\}, onT\_A_1 : \{Init\}, clear\_B_1 : \{Init\}, clear\_C_1 : \{Init\}, onT\_B_1 : \{Init\},$ $on\_C\_B_2 : \{BB\_C\_A\_B, \bot\}, clear\_A_2 : \{BB\_C\_A\_B, BT\_C\_A, \bot\}, onT\_C_2 : \{BT\_C\_A, \bot\},$ $on\_C\_A_2 : \{Persist, \bot\}, onT\_A_2 : \{Persist, \bot\}, clear\_B_2 : \{Persist, \bot\},$ $clear\_C_2 : \{Persist, \bot\}, on\_B\_C_2 : \{TB\_B\_C, \bot\}, onT\_B_2 : \{Persist, \bot\}$, ... $on\_A\_B_G : \{BB\_A\_C\_B, TB\_A\_B, Persist, \bot\}, on\_B\_C_G : \{TB\_B\_C, BB\_B\_A\_C, \bot\}$
Constraints:	**activity preconditions** $on\_A\_B\_G = TB\_A\_B \Rightarrow (onT\_A_3 \neq \bot \wedge clear\_A_3 \neq \bot \wedge clear\_B_3 \neq \bot)$ $on\_A\_B\_G = BB\_A\_C\_B \Rightarrow (on\_A\_C_3 \neq \bot \wedge clear\_A_3 \neq \bot \wedge clear\_B_3 \neq \bot)$ ...
Constraints:	**mutexes** $on\_C\_B_2 = BB\_C\_A\_B \Rightarrow onT\_C_2 \neq BT\_C\_A$ ...
Constraints:	**goal state** $on\_C\_B_G \neq \bot, on\_A\_B_G \neq \bot$

Table 22.3: Fragment of the generated CSP problem for 3 steps

If no solution is found, the planning graph is extended by adding an additional step and the CSP is extended accordingly until it is feasible. In the example, the CSP with 3 steps is feasible and a solution is: step 1: BT_C_A, step 2: TB_B_C, step 3: TB_A_B.

In [51], a slightly different CSP model is used with only boolean variables, one variable for each proposition in each step and one variable for each action in each step. Some redundant constraints (other than mutex) are identified and added to the CSP model. For instance the fact that an action cannot be immediately followed by its opposite action in a plan of optimal length.

In general, the reformulated problem is solved using classical CSP techniques (arc-consistency, dynamic variable ordering, memoization) and does not require specific constraint propagation techniques this is the reason why we will not focus on this family of approaches in the sequel of this chapter.

22.2.2 CSPs in Plan-Space Search

A second growing trend in planning is the extension of planning systems to reason about both time and resources. STRIPS is simply not expressive enough to represent more realistic planning problems. This demand for increased sophistication has led to the need for more powerful techniques to reason about time and resources during planning.

In Constraint-Based Planning, each search node represents a partial plan and consists of a set of time intervals, connected by constraints. The partial plan may be incomplete, in that constraints are not necessarily satisfied and pending choices have not been made. The planning process then involves modifying a partial plan until it has been turned into a complete and valid plan. Traditional search-based methods accomplish this by trying different options for completing partial plans, and backtracking when constraints are found to be violated. Constraint reasoning methods, such as propagation and consistency checks can be used to help out in that process.

The scheduling community has used constraint satisfaction techniques to perform this sort of reasoning. The main difference between constraint-based planning and scheduling is that, in planning, the set of activities of the plan is not completely known a-priori and must be determined during the search. The rules that govern the insertion of new activities in the plan are expressed as implicit or explicit constraints and, although they slightly differ from one planning system to the other [35, 42, 18, 40, 32], they all have the same flavor.

To fix the ideas, we will use in this section the IxTeT formalism [35, 42]. In this planning language the state of the world is described by a set of multi-valued state attributes together with a set of resource attributes. Each state attribute describes a particular feature of the world, for instance the position of a block. A resource is an object (or a set of objects) that can be simultaneously shared by several actions provided its maximal capacity is not exceeded. Operators (or tasks) are temporal structures composed of a set of events describing the changes of the world induced by the task (event), a set of assertions on state attributes describing conditions that must remain true during some time intervals (hold) and a set of resource usage (use, produce, consume) describing how the task uses some resources. All the above statements refer to time points that can be constrained with respect to the time interval [start,end] of the task. An example of model for the blocks world domain is shown in Table 22.4. Note that the domain is represented here with a finer grain than the STRIPS definition in Section 22.2.1: the operator is defined with internal time points, delays between these time points (duration) are expressed together with resource requirements (here, the task requires one hand from the two hands that are available to move the blocks). Beside resource usage, note that state attributes are very similar to state resources used in scheduling (see section 22.1.4) in that no task requiring a given state attribute to take different value can overlap in time.

constant	blocks = {a, b, c};
	positions = {a, b, c, table, hand};
attribute	clear(?x) { ?x in blocks; **?value in** {yes, no}; }
	on(?x) { ?x in blocks; **?value in** positions; }
resource	hands() { **capacity** = 2; }
task	TB(?x, ?y)(**start**, **end**) {
	// Move ?x from the table to atop ?y
	timepoint t1, t2;
	event(clear(?x):(yes,no), **start**);
	event(on(?x):(table,hand), t1);
	event(on(?x):(hand,?z), t2);
	event(clear(?z):(yes,no), t2);
	hold(on(?x):hand, (t1, t2));
	event(clear(?x):(no,yes), **end**);
	hold(clear(?x):no, (**start**, **end**));
	use(hands(): 1, (**start**, **end**));
	(t1 - **start**) **in** [00:10, 00:20];
	(t2 - t1) **in** [00:15, 00:25];
	(**end** - t2) **in** [00:10, 00:20];
	}

Table 22.4: Part of the blocks world domain in IxTeT

A partial plan is a set of tasks together with a set of constraints between the tasks variables (including temporal constraints between the task time points). The initial plan only contains a fake start task that asserts all the state attributes of the initial state and a fake end task that has the goal as (non-established) preconditions. A partial plan is complete and valid if and only if each instantiation of all the variables of the partial plan lead to a feasible fully-grounded plan where (1) the change of state attributes over time is non-ambiguously defined by and consistent with the events and assertions of the tasks of the plan and (2) resource constraints are satisfied. Figure 22.5 shows an example of a partial plan.

Partial plans are iteratively refined at each search node until the partial plan is complete and valid. Three types of plan refinements are considered:

- Non-established conditions are those events or assertions that are still not established by a task or the initial state. For instance, in the partial plan of Figure 22.5, the condition $clear(A) : yes$ related with the event at the start time point of task $TB(A, B)$ is still not established. Non-established conditions can be established by an existing event in the plan or by inserting a new task. In the case of the example, a new task $BT(C, A)$ could be added before the start of task $TB(A, B)$ to establish the condition.

- Possible conflicts between unordered events/assertions are pairs of statements that may require the same state attribute to take different values at the same moment.

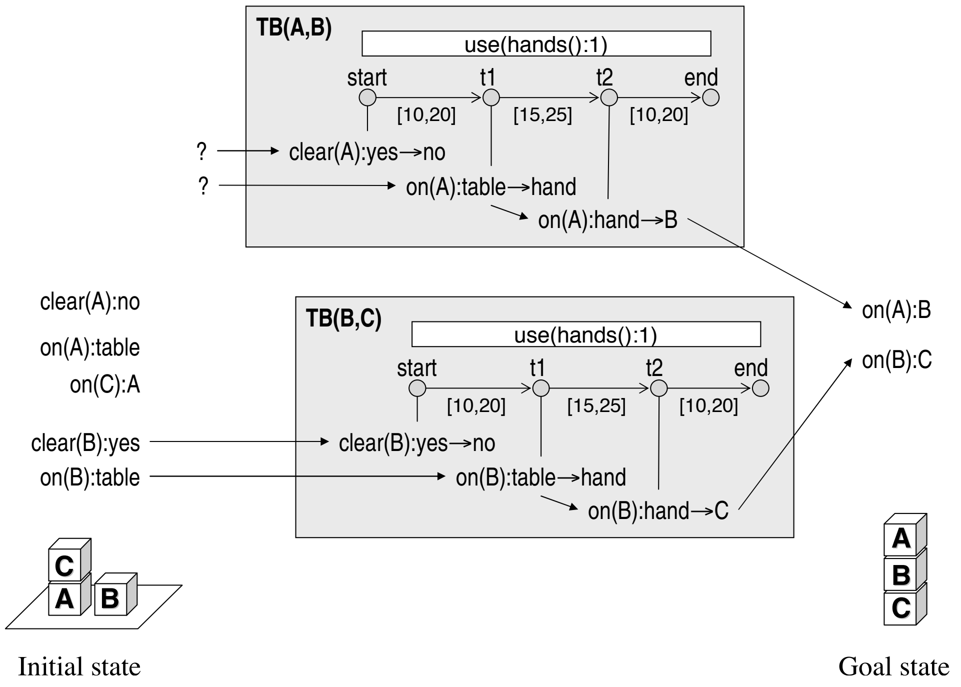


Figure 22.5: Partial plan

These incompatibility constraints can be solved by posting precedence constraints to order the conflicting events/assertions.

- Possible resource conflicts are subsets of resource requirements that may overlap in time and would in this case over-consume the resource. This would be the case of the two tasks in the example if the maximal capacity of the resource $hands$ is 1. These resource conflicts can be solved by ordering the tasks or, in case the resource can be produced, by inserting a task that produces the resource.

In Constraint-Based Planning, the partial plan is usually represented as a CSP with variables representing the task time points together with non-temporal variables appearing in the task definition as well as special variables representing the tasks that can be used to establish conditions in the partial plan. Specialized constraint propagation algorithms can be used to reduce the domain of variables or to deduce new temporal constraints. As far as time and resources are concerned, these propagation algorithms are essentially the same as the ones developed for pure scheduling problems, which will be described in Section 22.3. Indeed, unless stated otherwise, all the algorithms described in Section 22.3 can be implemented so as to accept other tasks and variables as the search evolves. We also sketch in that section how some of these algorithms can be adapted to propagate on state attributes. See [75] for recent advances in the field of using Constraint Programming in Plan-Space Search.

22.3 Constraint Propagation for Resource Constraints

Resource constraints represent the fact that activities require some amount of resource throughout their execution. In this section propagation for resource constraints is described for unary resources (machines), cumulative resources, and reservoirs. As we will see, the propagation of resource constraints is a purely deductive process that allows to deduce inconsistencies and to tighten the temporal characteristics of activities and resources. Throughout this section, we will concentrate on non-preemptive scheduling. We refer the reader to [5] for the generalization of the described constraint propagation techniques to preemptive and elastic scheduling.

22.3.1 Unary Resources

Several mechanisms have been developed to propagate unary resource constraints. Here we restrict the discussion to the disjunctive constraint propagation scheme and to the well-known edge-finding algorithm. Also note that the time tabling mechanism described in Section 22.3.2 can be applied to unary resources. We refer to [5] for a detailed introduction and comparison of several other constraint propagation techniques.

Disjunctive constraint propagation

Two activities A_i and A_j requiring the same unary resource cannot overlap in time. So, either A_i precedes A_j or A_j precedes A_i. If n activities require the resource, one thus has $n(n-1)/2$ (explicit or implicit) of such disjunctive constraints. Variants exist in the literature [24, 8, 44, 68, 74, 4], but in most cases the propagation consists of maintaining arc-B-consistency on the formula:

$$[\mathit{end}(A_i) \leq \mathit{start}(A_j)] \vee [\mathit{end}(A_j) \leq \mathit{start}(A_i)]$$

Enforcing arc-B-consistency on this formula is done as follows. Whenever the earliest end time of A_i exceeds the latest start time of A_j, A_i cannot precede A_j; hence A_j must precede A_i. The time bounds of A_i and A_j are consequently updated with respect to the new temporal constraint $\mathit{end}(A_j) \leq \mathit{start}(A_i)$. Similarly, when the earliest end time of A_j exceeds the latest start time of A_i, A_j cannot precede A_i. When neither of the two activities can precede the other, a contradiction is detected.

In Constraint-Based Planning, the disjunctive constraint can easily be adapted to propagate conflicts between mutually exclusive statements [75].

Edge-finding

The edge-finding constraint propagation technique consists of deducing that some activities from a given set Ω must, can, or cannot, execute first (or last) in Ω. Such deductions lead to new ordering relations ("edges" in the graph representing the possible orderings of activities) and new time bounds, *i.e.*, strengthened earliest start times and latest end times of activities.

In the following, r_Ω, d_Ω, and p_Ω denote the smallest of the earliest start times, the largest of the latest end times, and the sum of the minimal processing times of the activities in Ω, respectively. Let $A_i \ll A_j$ ($A_i \gg A_j$) mean that A_i executes before (after) A_j and

$A_i \ll \Omega$ ($A_i \gg \Omega$) mean that A_i executes before (after) all the activities in Ω. Once again, variants exist [9, 10, 11, 12, 54, 7, 52, 59, 77] but the following rules capture the "essence" of the edge-finding propagation technique:

$$\forall_\Omega\ \forall_{A_i \notin \Omega}\ [d_{\Omega \cup \{A_i\}} - r_\Omega < p_\Omega + p_i] \Rightarrow [A_i \ll \Omega]$$
$$\forall_\Omega\ \forall_{A_i \notin \Omega}\ [d_\Omega - r_{\Omega \cup \{A_i\}} < p_\Omega + p_i] \Rightarrow [A_i \gg \Omega]$$
$$\forall_\Omega\ \forall_{A_i \notin \Omega}\ [A_i \ll \Omega] \Rightarrow [\mathit{end}(A_i) \leq \min_{\emptyset \neq \Omega' \subseteq \Omega} (d_{\Omega'} - p_{\Omega'})]$$
$$\forall_\Omega\ \forall_{A_i \notin \Omega}\ [A_i \gg \Omega] \Rightarrow [\mathit{start}(A_i) \geq \max_{\emptyset \neq \Omega' \subseteq \Omega} (r_{\Omega'} + p_{\Omega'})]$$

If n activities require the resource, there are a priori $O(n * 2^n)$ pairs (A_i, Ω) to consider. Still, it is easy to see that when a rule is applied for some set Ω, the same rule provides even better deductions for the superset $\Omega' = \{A_i \mid [r_i, d_i) \subseteq [r_\Omega, d_\Omega)\}$. This makes that one can restrict the rules to sets Ω_I made of all activities whose time window belongs to some interval I. As there are no more that $O(n^2)$ such intervals I, we have a straightforward polynomial algorithm, running in cubic time, to implement the edge-finding rules. Algorithms that perform all the time bound adjustments in $O(n^2)$ are presented in [10, 56, 54]. Another variant of the edge-finding technique is presented in [11]. It runs in $O(n \log n)$ but requires much more complex data structures. [77] presents another variant running in $O(n \log n)$ that requires less complex data structures than the ones used in [11].

Techniques similar to edge-finding have been proposed to propagate groups of mutex relations in planning [75].

"Not-first" and "not-last" rules

"Not-first" and "not-last" propagation rules have also been developed as a "negative" counterpart to edge-finding. These rules deduce that an activity A_i cannot be the first (or the last) to execute in $\Omega \cup \{A_i\}$.

$$\forall_\Omega\ \forall_{A_i \notin \Omega}\ [d_{A_i} - r_\Omega < p_\Omega + p_i] \Rightarrow [\mathit{end}(A_i) \leq \max_{B \in \Omega} \mathit{lst}_B]$$
$$\forall_\Omega\ \forall_{A_i \notin \Omega}\ [d_\Omega - r_{A_i} < p_\Omega + p_i] \Rightarrow [\mathit{start}(A_i) \geq \min_{B \in \Omega} \mathit{eet}_B]$$

The corresponding time bound adjustments can be made in $O(n^2)$ [3, 71].

Conjunctive reasoning between temporal and resource constraints

The above given propagation techniques reason on the time bounds of activities on one unary resource. In [57, 69] propagation techniques are presented that reason on the combination of time bounds of activities on multiple unary resources and the temporal constraints linking these activities. Even though these techniques have led to good computational results, they have not yet been studied much. Propagation techniques that reason on the combination of activity time bounds and temporal constraints on one cumulative resource or reservoir have been studied. We discuss these techniques in the Section 22.3.3.

22.3.2 Cumulative Resources

Cumulative resource constraints represent the fact that activities A_i use some amount $cap(A_i)$ of resource throughout their execution. Many algorithms have been proposed for the propagation of the non-preemptive cumulative constraint. A limited subset of these algorithms is presented in this section. In the remainder of this chapter, c_i denotes the minimal value of $cap(A_i)$, *i.e.*, the minimal capacity required by A_i.

Timetable constraint

First we consider the timetable mechanism, widely used in Constraint-Based Scheduling tools, that allows to propagate the resource constraint in an incremental fashion. The "timetable" is used to maintain information about resource utilization and resource availability over time. Resource constraints are propagated in two directions. From resources to activities, to update the time bounds of activities (earliest start times and latest end times) according to the availability of resources; and from activities to resources, to update the timetables according to the time bounds of activities. Although several variants exist [44, 31, 45, 67, 50] the propagation mainly consists of maintaining for any time t arc-B-consistency on the formula:

$$\sum_{A_i \mid \mathit{start}(A_i) \leq t < \mathit{end}(A_i)} \mathit{cap}(A_i) \leq \mathit{cap}(R)$$

Disjunctive constraint

Let A_i and A_j be two activities such that $c_i + c_j > \mathit{cap}(R)$. As such they cannot overlap in time and thus either A_i precedes A_j or A_j precedes A_i, *i.e.*, the disjunctive constraint holds between these activities. In general the disjunctive constraint achieves arc-B-consistency on the formula

$$[\mathit{cap}(A_i) + \mathit{cap}(A_j) \leq \mathit{cap}(R)] \vee [\mathit{end}(A_i) \leq \mathit{start}(A_j)] \vee [\mathit{end}(A_j) \leq \mathit{start}(A_i)]$$

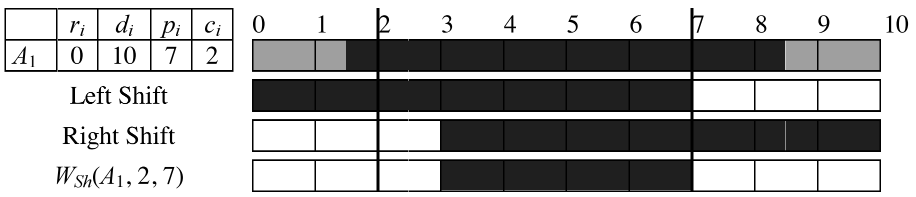
Energy reasoning

Energy based constraint propagation algorithms compare the amount of energy provided by a resource over some interval $[t_1, t_2)$ to the amount of energy required by activities that have to be processed over this interval. [25] proposes the following definition of the required energy consumption that takes into account the fact that activities cannot be interrupted. Given an activity A_i and a time interval $[t_1, t_2)$, $W_{Sh}(A_i, t_1, t_2)$, the "Left-Shift / Right-Shift" required energy consumption of A_i over $[t_1, t_2)$ is c_i times the minimum of the three following durations.

- $t_2 - t_1$, the length of the interval;
- $p_i^+(t_1) = \max(0, p_i - \max(0, t_1 - r_i))$, the number of time units during which A_i executes after time t_1 if A_i is left-shifted, *i.e.*, scheduled as soon as possible;

- $p_i^-(t_2) = \max(0, p_i - \max(0, d_i - t_2))$, the number of time units during which A_i executes before time t_2 if A_i is right-shifted, *i.e.*, scheduled as late as possible.

This leads to $W_{Sh}(A_i, t_1, t_2) = c_i \min(t_2 - t_1, p_i^+(t_1), p_i^-(t_2))$. In Figure 22.6 an example is given where the required energy consumption of A_1 over $[2, 7)$ is 8. Indeed, at least 4 time units of A_1 have to be executed in $[2, 7)$; *i.e.*, $W_{Sh}(A, 2, 7) = 2\min(5, 5, 4) = 8$.



	r_i	d_i	p_i	c_i
A_1	0	10	7	2

Figure 22.6: Left-Shift / Right-Shift.

The Left-Shift / Right-Shift overall required energy consumption $W_{Sh}(t_1, t_2)$ over an interval $[t_1, t_2)$ is then defined as the sum over all activities A_i of $W_{Sh}(A_i, t_1, t_2)$. The Left-Shift / Right-Shift slack $S_{Sh}(t_1, t_2)$ over $[t_1, t_2)$ is defined as $C(t_2 - t_1) - W_{Sh}(t_1, t_2)$. It is obvious that if there is a feasible schedule then $S_{Sh}(t_1, t_2) \geq 0$ for all t_1 and t_2 such that $t_2 \geq t_1$.

It is shown in [5] how the values of W_{Sh} can be used to adjust activity time bounds. Given an activity A_i and a time interval $[t_1, t_2)$ with $t_2 < d_i$, it is examined whether A_i can end before t_2. If there is a time interval $[t_1, t_2)$ such that

$$W_{Sh}(t_1, t_2) - W_{Sh}(A_i, t_1, t_2) + c_i p_i^+(t_1) > C(t_2 - t_1)$$

then a valid lower bound of the end time of A_i is

$$t_2 + \frac{1}{c_i}(W_{Sh}(t_1, t_2) - W_{Sh}(A_i, t_1, t_2) + c_i p_i^+(t_1) - C(t_2 - t_1))$$

Similarly, when

$$W_{Sh}(t_1, t_2) - W_{Sh}(A_i, t_1, t_2) + c_i \min(t_2 - t_1, p_i^+(t_1)) > C(t_2 - t_1),$$

A_i cannot start before t_1 and a valid lower bound of the start time of A_i is

$$t_2 - \frac{1}{c_i}(C(t_2 - t_1) - W_{Sh}(t_1, t_2) + W_{Sh}(A_i, t_1, t_2)).$$

[5] presents a $O(n^3)$ algorithm to compute these time bound adjustments for all n activities. It is first shown there are $O(n^2)$ intervals $[t_1, t_2)$ of interest. Given an interval and an activity, the adjustment procedure runs in $O(1)$. As such the obvious overall complexity of the algorithm is thus $O(n^3)$. An interesting open question is whether there is a quadratic algorithm to compute all the adjustments on the $O(n^2)$ intervals under consideration. Another open question at this point is whether the characterization of the $O(n^2)$

time intervals in [5] can be sharpened in order to eliminate some intervals and reduce the practical complexity of the corresponding algorithm. Finally, it seems reasonable to think that the time bound adjustments could be sharpened. Even though the energy tests can be limited (without any loss) to a given set of intervals, it could be that the corresponding adjustment rules cannot.

22.3.3 Conjunctive Reasoning between Temporal and Resource Constraints

The propagation algorithms described in the previous section reason on the time bounds of activities (r_i, lst_i, eet_i,d_i) and do not directly take into account the precedence constraints that may exist between them. We describe in this section two recently proposed propagation algorithms respectively on cumulative resources and reservoirs. Like previous propagation algorithms, both of them are used to discover new time bounds and/or new precedence relations. The main originality lies in the fact that they analyze the relative position of activities (precedence relations in the precedence graph) rather than their absolute position only, as was the case for the previously discussed propagation techniques. As a consequence, they allow a much stronger propagation when the time windows of activities are large and when the current schedule contains a lot of precedence relations, which is typically the case when integrating planning and scheduling.

Precedence graph

The algorithms presented in this section require that a temporal network representing the relations between the time points of all activities (start and end times) using the Point Algebra [76] is maintained during search. We denote the set of qualitative relations between time points by $\{\emptyset, \prec, \preceq, =, \succ, \succeq, \neq, ?\}$. The temporal network is in charge of maintaining the transitive closure of those relations.

If s_i and e_i denote the start and end time-point of activity A_i, the initial set of relations consist of the precedences $s_i \prec e_i$ for each activity A_i and $x_i \preceq x_j$ or $x_i \prec x_j$ for each precedence constraint $x_i \leq x_j + d_{ij}$ depending on the value of d_{ij}. For instance, for a precedence constraint $\mathit{end}(A_i) \leq \mathit{start}(A_j)$ between two activities A_i and A_j, the initial precedence graph contains the relation $e_i \preceq s_j$.

During search additional precedence relations can be added as decisions or as the result of constraint propagation.

Energy precedence constraint

The *energy precedence* propagation [41] for an activity A_i on a cumulative resource R_k ensures that for each subset ϕ of predecessor activities of activity A_i the resource provides enough energy to execute all activities in ϕ between r_ϕ and s_i. More formally, it performs the following deduction rule

$$\forall_{\phi \subseteq \{A_j | e_j \preceq s_i\}} \ \mathit{start}(A_i) >= r_\phi + \lceil E_\phi / cap(R_k) \rceil$$

where E_ϕ is the sum of the minimal value of $E(A_j, R_k)$ (minimal energy) over all A_j in ϕ. The propagation of the energy precedence constraint can be performed for all the activities on a resource and for all the subsets ϕ with a total worst-case time complexity of $O(n(p + log(n))$ where n is the number of activities on the resource and p the maximal number of predecessors of a given activity in the temporal network ($p < n$).

Balance constraint

On a reservoir resource, if x is the start or end time of an activity that changes the reservoir level, we denote by $q(x)$ the integer variable representing the relative change of the reservoir level due to the activity. By convention, $0 < q(x)$ represents a production event and $q(x) < 0$ a consumption event. The basic idea of the *balance constraint* [41] is to for each event x compute an upper and lower bound on the reservoir level at the time of x. Using the temporal network, an upper bound on the reservoir level at date $x - \epsilon$ just before x can be computed assuming that:

- All the production events y that *may* be executed strictly before x are executed strictly before x and produce as much as possible, *i.e.*, produce $ub(q(y))$ (denoted by $q_{max}(y)$). Let $Poss^{\preceq}(x) = \{y \mid \neg(x \prec y)\}$ denote the set of events that may be executed before or at the same time as x.
- All the consumption events y that *need* to be executed strictly before x are executed strictly before x and consume as little as possible, *i.e.*, consume $q_{max}(y)$[1]. Let $Nec^{\prec}(x) = \{y \mid (y \prec x)\}$ denote the set of events that necessarily execute strictly before x.
- All the consumption events that *may* execute simultaneously or after x are executed simultaneously or after x.

More formally, if P is the set of production events and C the set of consumption events, the upper bound can be computed as follows:

$$L^{<}_{max}(x) = \sum_{y \in P \cap Poss^{\preceq}(x)} q_{max}(y) + \sum_{y \in C \cap Nec^{\prec}(x)} q_{max}(y) \tag{22.2}$$

For symmetry reasons, we describe only the propagation based on $L^{<}_{max}(x)$. Using this bound, the balance constraint is able to discover four types of information: dead ends, new bounds for required capacity variables, new bounds for time variables, and new precedence relations.

- Discovering dead ends. This is the most trivial propagation. Whenever $L^{<}_{max}(x) < 0$, we know that the level of the reservoir will surely be negative just before event x so the search has reached a dead end.
- Discovering new bounds on required capacity variables. Suppose there exists a consumption event $y \in Nec^{\prec}(x)$ such that $q_{max}(y) - q_{min}(y) > L^{<}_{max}(x)$. If y would consume a quantity q such that $q_{max}(y) - q > L^{<}_{max}(x)$ then, simply by replacing $q_{max}(y)$ by q in (22.2), we see that the level of the reservoir would be negative just before x. Thus, we can deduce that $q(y) \geq q_{max}(y) - L^{<}_{max}(x)$.
- Discovering new bounds on time variables.

  $\backslash Nec^{\prec}(x)$ consists of the set of events that may but need not necessarily execute strictly before x, *i.e.*, they can also execute at the same time as or after x. Let $P(x)$

[1]For a consumption event, $q < 0$ and thus, q_{max} really corresponds to the smallest consumption of the event.

denote the set of production events in $Poss^{\preceq}(x) \setminus Nec^{\prec}(x)$. The reasoning behind the deduction here is that if the maximal reservoir level of all events necessarily executed strictly before x is negative, then some production events in $P(x)$ need to be scheduled before x, more specifically x needs to be scheduled after a sufficient number of these production events to not have a negative reservoir level. More formally, let's rewrite (22.2) as follows:

$$L^{<}_{max}(x) = \sum_{y \in Nec^{\prec}(x)} q_{max}(y) + \sum_{y \in P \cap (Poss^{\preceq}(x) \setminus Nec^{\prec}(x))} q_{max}(y)$$

The first term of this equation is the sum of the maximal production and minimal consumption of the events that necessarily execute strictly before x, thus giving the maximal reservoir level. As $L^{<}_{max}(x) \geq 0$, we know that if this term is negative, it means that some production events in $P(x)$ will have to be executed strictly before x in order to produce at least:

$$\Delta^{<}_{min}(x) = - \sum_{y \in Nec^{\prec}(x)} q_{max}(y)$$

We suppose the production events $(y_1, \cdots, y_i, \cdots, y_p)$ in $P(x)$ are ordered by non-decreasing minimal time $t_{min}(y)$. $t_{min}(y)$ is either the earliest start time or earliest end time, depending on whether y is a start or end time. Let k be the index in $[1, p]$ such that:

$$\sum_{i=1}^{k-1} q_{max}(y_i) < \Delta^{<}_{min}(x) \leq \sum_{i=1}^{k} q_{max}(y_i)$$

If event x is executed at a date $t(x) \leq t_{min}(y_k)$, not enough producers will be able to execute strictly before x in order to ensure a positive level just before x. Thus, $t_{min}(y_k) + 1$ is a valid lower bound of $t(x)$.

- Discovering new precedence relations. There are cases where one can perform an even stronger propagation. $P(x)$ is again the set of production events in $Poss^{\preceq}(x) \setminus Nec^{\prec}(x)$. If there is one production event y in $P(x)$ that is needed to produce before x to not get a negative reservoir level, a precedence relation can be deduced between y and x. So, suppose there exists a production event y in $P(x)$ such that:

$$\sum_{z \in P(x) \cap Poss^{\preceq}(y)} q_{max}(z) < \Delta^{<}_{min}(x)$$

Then, if we had $t(x) \leq t(y)$, we would see that again there is no way to produce $\Delta^{<}_{min}(x)$ before event x as the only events that could produce strictly before event x are the ones in $P(x) \cap Poss^{\preceq}(y)$. Thus, we can deduce the necessary precedence relation: $t(y) < t(x)$. Note that a weaker version of this propagation has been proposed in [16] that runs in $O(n^2)$ and does not analyze the precedence relations between the events of $P(x)$.

The balance algorithm can be executed for all the events x on a reservoir with a global worst-case complexity in $O(n^2)$ if the propagation that discovers new precedence relations is not turned on, and in $O(n^3)$ for a full propagation. In practice, there are many ways to shortcut this worst case and in particular, it has been noticed that the algorithmic cost of the extra propagation that discovers new precedence relations was in general negligible. Unlike all the other propagation algorithms we have seen in this section 22.3, the balance constraint cannot directly be applied in planning problems because it assumes that the set or producer and consumer events is completely known. The extension of the balance constraint to planning problems is discussed in [41].

22.4 Constraint Propagation on Optimization Criteria

As said in Section 22.1.5, the commonly used way of modeling an objective function is by introducing a variable *criterion* that is constrained to be equal to the value of the objective function. In cases where the objective function F is a function of the end variables of the activities and F is a maximum such as $C_{\max}$ or $T_{\max}$, considering the objective constraint and the resource constraints independently is not a problem. Indeed, the upper bound on *criterion* is directly propagated on the end time of each activity, *i.e.*, latest end times are tightened efficiently.

The situation is more complex for sum functions such as $\sum w_i C_i$, $\sum w_i T_i$, or $\sum w_i U_i$, and for objective functions that are not a function of the end variables of the activities like sum of setup times and sum of setup costs. For several of these cases, dedicated constraint propagation techniques have been developed often taking the resource constraints and the objective function simultaneously into account.

In this section we describe such dedicated constraint propagation techniques for two objective functions in more detail: weighted number of late activities ($\sum w_i U_i$) and sum of setup times and setup costs. For more general considerations on cost-based constraint propagation we refer to [28].

22.4.1 Weighted Number of Late Activities

In this section we pay attention to constraint propagation for the objective function $\sum w_i U_i$, i.e., minimizing the weighted number of late activities, as described in [5].

The basis for this constraint propagation is formed by calculating a good lower bound on the weighted number of late activities. Such a lower bound is obviously also a lower bound for the variable *criterion*. Relaxing non-preemption is a well-known technique to obtain good lower bounds in scheduling. Unfortunately, the preemptive problem remains NP-Hard. A "relaxed preemptive lower bound", *i.e.*, a slightly stronger relaxation than the preemptive relaxation, can be used. As explained below, it can be computed in $O(n^2 \log n)$.

Let us recall a well-known result for the One-Machine Scheduling Problem (*i.e.*, the problem of scheduling activities on a unary resource). Its preemptive relaxation is polynomial and has the very interesting property that there exists a feasible preemptive schedule if and only if over any interval $[t_1, t_2)$, the sum of the processing times of the activities in $\{A_i \mid [t_1 \le r_i] \wedge [d_i \le t_2]\}$ is at most $t_2 - t_1$. It is well-known that relevant values for t_1 and t_2 are respectively the release dates and the deadlines [8].

A decision variable x_i per activity is introduced that equals 1 when the activity is on-time and 0 otherwise. Notice that if $d_i \leq \delta_i$, A_i is on-time in any solution, *i.e.*, $x_i = 1$. In such a case we adjust the value of δ_i to d_i (this has no impact on solutions) so that due dates are always smaller than or equal to deadlines. We also assume that there is a preemptive schedule that meets all deadlines (if not, the resource constraint does not hold and a backtrack occurs). The following Mixed Integer Program (MIP) [64] computes the minimum weighted number of late activities in the preemptive case:

$$\min \sum_{1}^{n} w_i(1 - x_i)$$
$$u.c. \begin{cases} \forall_{t_1} \forall_{t_2 > t_1} \sum\limits_{S(t_1,t_2)} p_i + \sum\limits_{P(t_1,t_2)} p_i x_i \leq t_2 - t_1 \\ \forall_{i \in \{1,\dots,n\}} \; x_i \in \{0,1\} \end{cases} \tag{22.3}$$

where $S(t_1, t_2)$ is the set of activities that are sure to execute between t_1 and t_2 and where $P(t_1, t_2)$ is the set of activities that are preferred to execute between t_1 and t_2.

$$\begin{aligned} S(t_1, t_2) &= \{A_i \mid r_i \geq t_1 \wedge d_i \leq t_2\} \\ P(t_1, t_2) &= \{A_i \mid r_i \geq t_1 \wedge d_i > t_2 \; \wedge \delta_i \leq t_2\} \end{aligned}$$

Actually, it is easy to see that the relevant values of t_1 correspond to the release dates and those of t_2 to the due dates and deadlines. Hence, there are $O(n^2)$ constraints in the MIP. We now focus on the *continuous relaxation* of (22.3) in which for any activity A_i such that $r_i + p_i > \delta_i$, *i.e.*, for any late activity, the constraint $x_i = 0$ is added.

$$\min \sum_{1}^{n} w_i(1 - x_i)$$
$$u.c. \begin{cases} \forall_{t_1 \in \{r_i\}} \forall_{t_2 \in \{d_i\} \cup \{\delta_i\} \mid t_2 > t_1} \sum\limits_{S(t_1,t_2)} p_i + \sum\limits_{P(t_1,t_2)} p_i x_i \leq t_2 - t_1 \\ \forall_i \; r_i + p_i > \delta_i \Rightarrow x_i = 0 \\ \forall_{i \in \{1,\dots,n\}} \; x_i \in [0,1] \end{cases} \tag{22.4}$$

The linear program (22.4) can be solved with an LP solver and we can use reduced costs to prove that some activities can, must or cannot end before their due date. In [5] an $O(n^2 \log n)$ algorithm is described solving the same problem.

22.4.2 Sum of Setup Times and Sum of Setup Costs

In this section we pay attention to constraint propagation of setup time and setup cost constraints. We discuss the constraint propagation as described in [30]. We also refer to [29, 28] that extend work of Brucker and Thiele [7] in the context of CP. To simplify the presentation, we only consider the case where there are no cumulative resources. We do include the possible presence of alternative resources (see Section 22.1.4).

The basis for the constraint propagation of setup times and setup costs described in this section is formed by using a *routing problem* as a relaxation of the scheduling problem. In this problem, one has a set of *start* nodes, a set of *internal* nodes, and a set of *end* nodes. Each internal node i represents an activity A_i. When having m alternative machines, one is looking for m disjoint *routes* or *paths* in the graph defined by these three sets. Each route corresponds to a different machine, starting in the start node of the machine, traversing a sequence of internal nodes, and ending in the end node of the machine. More precisely, let $I = \{1, \ldots, n\}$ be a set of n nodes, and $E = \{n+1, \ldots, n+m\}$ and $S = \{n+m+1, \ldots, n+2*m\}$ two sets of m nodes. Nodes in I represent internal nodes, nodes in S represent start nodes, and nodes in E represent end nodes. A global constraint is defined ensuring that m different routes $\rho_1, \ldots, \rho_m$ exist such that all internal nodes are visited exactly once by a route starting from a node in S and ending in a node in E. Start nodes $n+m+1, \ldots, n+2*m$ belong to routes $\rho_1, \ldots, \rho_m$, respectively. End nodes $n+1, \ldots, n+m$ belong to routes $\rho_1, \ldots, \rho_m$, respectively. Moreover, sets of possible routes can be associated to each internal node.

In the CP model three variables per node are defined. Variables next_i and prev_i identify the nodes visited directly after and directly before node i, respectively. Variables route_i identify the route node i belongs to. Variables next_i and prev_i take their values in $\{1, \ldots, n+2m\}$. Variables route_i take their values in $\{1, \ldots, m\}$. Each start and end node has its route variable bound, *i.e.*, $\mathit{route}_{n+1} = 1, \ldots, \mathit{route}_{n+m} = m$, $\mathit{route}_{n+m+1} = 1$, $\ldots, \mathit{route}_{n+2m} = m$. In order to have a uniform treatment of all nodes inside the constraint, each start node $n+m+u$ has its prev_{n+m+u} variable bound to the corresponding end node ($\mathit{prev}_{n+m+u} = n+u$), and each end node $n+u$ has its next_{n+u} variable bound to the corresponding start node ($\mathit{next}_{n+u} = n+m+u$). There furthermore exists a setup cost c^u_{ij} that expresses that if node j is visited directly after node i on a route u ($\mathit{next}_i = j, \mathit{route}_i = \mathit{route}_j = u$), a cost c^u_{ij} is induced. A feasible solution is defined as an assignment of distinct values to each next variable, while avoiding sub-tours (tours containing only internal nodes), and respecting the constraints

$$\begin{aligned} \mathit{next}_i = j \quad &\Leftrightarrow \quad \mathit{prev}_j = i \\ \mathit{next}_i = j \quad &\Rightarrow \quad \mathit{route}_i = \mathit{route}_j \end{aligned}$$

The problem is then to find an optimal feasible solution, *i.e.*, a feasible solution that minimizes

$$\sum_{i=1}^{n} c^u_{i\,next_i} \tag{22.5}$$

As said, the routing problem described constitutes a relaxation of the global scheduling problem. If an internal node i has its next variable assigned to another internal node j, activity A_i directly precedes activity A_j. If an internal node i has its next variable assigned to an ending node $n+u$, activity A_i is the last activity scheduled on machine R_u. The setup cost function c^u_{ij} of the routing problem corresponds to the setup times $\mathit{setup}(A_i, A_j, R_u)$ or setup costs $\mathit{setupCost}(A_i, A_j, R_u)$ between activities (see Section 22.1.4). As such the minimization of the total setup cost (22.5) in the routing problem corresponds to the minimization of the sum of setup times or setup costs in the scheduling problem.

Route optimization constraint

One of the basic ideas of the constraint propagation in [30] is to create a global constraint having a propagation algorithm aimed at removing those assignments from variable domains which do not improve the best solution found so far. Domain reduction is achieved by optimally solving an Assignment Problem [22] which is a relaxation of the routing problem described and thus also of the global scheduling problem. The Assignment Problem is the graph theory problem of finding a set of disjoint sub-tours such that all the vertices in a graph are visited and the overall cost is minimized.

In the routing problem we look for a set of m disjoint routes each of them starting from a start node and ending in the corresponding end node covering all nodes in a graph, *i.e.*, considering that each end node is connected to the corresponding start node, we look for a set of m disjoint tours each of them containing a start node. This problem can be formulated as an Assignment Problem on the graph defined by the set of nodes in the routing problem and the set of arcs (i, j) such that $j \in \textit{domain}(\textit{next}_i)$. The cost on arc (i, j) is the minimal setup cost (or time), *i.e.*,

$$\min_{u \in \textit{domain}(\textit{route}_i) \cap \textit{domain}(\textit{route}_j)} \textit{setupCost}(A_i, A_j, R_u).$$

The value of the optimal solution of the Assignment Problem is obviously a lower bound on the value of the optimal solution of the routing problem. The *primal-dual* algorithm described in [34] provides an optimal integer solution for the Assignment Problem. Besides this optimal assignment with the corresponding lower bound LB on the original problem, a reduced cost matrix $\bar{c}$ is obtained. Each $\bar{c}_{ij}$ estimates the additional cost to be added to LB if variable $\textit{next}_i$ takes the value j. These results can be used both in constraint propagation as in the definition of search heuristics. The lower bound LB is trivially linked to the *criterion* variable representing the objective function through the constraint $LB \leq \textit{criterion}$. More interesting is the propagation based on reduced costs. Given the reduced cost matrix $\bar{c}$, it is known that $LB_{\textit{next}_i = j} = LB + \bar{c}_{ij}$ is a valid lower bound for the problem where $\textit{next}_i$ takes the value j. Therefore we can impose

$$LB_{\textit{next}_i = j} > \textit{ub}(\textit{criterion}) \Rightarrow \textit{next}_i \neq j$$

For more details on the use of reduced costs for setup constraints we refer to [30]. We remark that reduced cost fixing appears to be particularly suited for CP. In fact, while reduced cost fixing is extensively used in OR frameworks, it is usually not exploited to trigger other constraints, but only in the following lower bound computation, *i.e.*, the following node in the search tree. When embedded in a CP framework, the reduced cost fixing produces domain reductions which usually trigger propagation from other constraints in the problem through shared variables.

Precedence graph constraint

Linking the routing model and the scheduling model is done thanks to a *precedence graph constraint*. This constraint maintains for each machine R_u an extended precedence graph G_u that allows to represent and propagate temporal relations between pairs of activities on the machine as well as to dynamically compute the transitive closure of those relations. More precisely, G_u is a graph whose vertices are the alternative activities A_i^u that may execute on machine R_u (see Section 22.1.4). A node A_i^u is said to *surely contribute* if machine R_u is the only possible machine on which A_i can be processed. Otherwise, if activity A_i can also be processed on other machines, the node A_i^u is said to *possibly contribute*. Two kinds of edges are represented on G_u:

- A *precedence edge* between two alternative activities $A_i^u \rightarrow A_j^u$ means that if machine R_u is chosen for both activities A_i and A_j, then A_j will have to be processed after A_i on R_u.
- A *next edge* between two alternative activities $A_i^u \Rightarrow A_j^u$ means that if machine R_u is chosen for both activities A_i and A_j then A_j will have to be processed directly after A_i on R_u. No activity may be processed on R_u between A_i and A_j.

The first role of the precedence graph is to incrementally maintain the closure of this graph when new edges or vertices are inserted, *i.e.*, to deduce new edges given the ones already present in the graph. The following five rules [30] are used by the precedence graph:

1. If $A_i^u \rightarrow A_j^u$, $A_j^u \rightarrow A_i^u$, and A_i^u surely contributes then A_j^u does not contribute (Incompatibility rule).
2. If $A_i^u \rightarrow A_l^u$, $A_l^u \rightarrow A_j^u$, and A_l^u surely contributes then $A_i^u \rightarrow A_j^u$ (Transitive closure through contributor).
3. If $A_l^u \Rightarrow A_i^u$, $A_l^u \rightarrow A_j^u$, and A_l^u surely contributes then $A_i^u \rightarrow A_j^u$ (Next-edge closure on the left).
4. If $A_j^u \Rightarrow A_l^u$, $A_i^u \rightarrow A_l^u$, and A_l^u surely contributes then $A_i^u \rightarrow A_j^u$ (Next-edge closure on the right).
5. If for all A_l^u either $A_l^u \rightarrow A_i^u$ or $A_j^u \rightarrow A_l^u$ then $A_i^u \Rightarrow A_j^u$ (Next-edge finding).

New edges are added on the precedence graph G_u by the scheduling constraints (precedence and resource constraints) and by the route optimization constraint (whenever a variable *next*$_i$ is bound a new next-edge is added). Besides computing the incremental closure, the precedence graph also incrementally maintains the set of activities that are possibly next to a given activity A_i^u given the current topology of G_u. As such it allows to effectively reduce the domain of the variables *next*$_i$ and *prev*$_i$. Furthermore, the precedence graph constraint propagates the current set of precedence relations expressed on G_u on the start and end variables of activities.

22.5 Heuristic Search

The general principles around search in CP apply to both the planning and scheduling domain:

- Since for complexity reasons constraint propagation cannot remove all impossible values from the domains of variables, heuristic search is required to generate a solution to the problem instance under consideration.
- Once a solution with a given cost is found, this heuristic search can be either continued or restarted with an additional constraint stating that only solutions with a lower cost are searched for. In the case of multiple criteria, this additional constraint can be replaced by a set of constraints authorizing the solution to deteriorate for some criteria if it improves for others.

- Some variables are more constrained than others, depending on the problem instance: some activities lie on a critical path of the precedence graph, some resources are more heavily loaded than others, etc. Focusing on the more constrained variables first is more likely to quickly lead to a solution.

However, the significance of temporal and resource constraints makes it possible to use these principles in domain-specific manners. Let us first consider the case of the pure Job Shop Scheduling Problem [33]. The variables of the problem are basically just the start and end times of activities and the *criterion* variable representing the makespan ($C_{\max}$). Temporal constraints relating these variables are propagated in a perfect manner, *i.e.*, the earliest and latest start and end times resulting from constraint propagation guarantee that the temporal constraints are satisfied. The only remaining constraints are the resource constraints. As there are only unary resources, no two activities A_i and A_j requiring the same resource can overlap in time, *i.e.*, either A_i precedes A_j or A_j precedes A_i (see Section 22.3.1). Following this basic observation, rather than attempting to instantiate the start and end variables, an appealing and often much more efficient strategy consists in deciding in which order activities shall execute, *i.e.*, whether A_i shall execute before A_j or A_j before A_i.

Although it is less immediate, the same type of branching strategy can also be considered for cumulative resources. Indeed, whenever n non-preemptible activities are such that the sum of the capacities required exceeds the available capacity (for a given resource), at least two of these activities cannot overlap in time, and hence must be ordered. An alternative but equivalent view consists in considering a cumulative resource R of capacity $cap(R)$ as a set of $cap(R)$ "lines" of capacity 1, on which activities cannot overlap. Hence, if the activities can be organized along at most c sequences such that (i) an activity A_i requiring capacity $cap(A_i)$ appears in $cap(A_i)$ sequences and (ii) activities in each sequence are totally ordered by temporal constraints, then the satisfaction of the temporal constraints guarantees the satisfaction of the resource constraint.

In practice, the right branching strategy also depends on the optimization criterion (or multiple criteria) to optimize:

- An optimization criterion to minimize is called "regular" if it increases with the end times of the activities. In other terms, a solution S cannot be strictly better than another solution S' if no activity A_i finishes earlier in S than in S'. Examples of regular criteria include the makespan, the average completion time of the activities, the maximal or weighted tardiness of activities, the weighted number of late activities. When the optimization criterion is regular, it is particularly appropriate to solve the resource constraints by ordering activities: on any given branch of the search tree, the value of the criterion obtained by replacing each end time variable by its lower bound is a lower bound for the optimization function. In addition, if at a given node the earliest start and end times satisfy all the constraints of the problem (which is the case for resource constraints if they have been replaced by appropriate temporal constraints and these temporal constraints have been propagated), then these earliest start and end times provide the best solution attainable from this node. "Dominance properties" can also be applied to prune some nodes: whenever it can be shown that for any schedule attainable from a node, an equivalent or better schedule is attainable from another node, the first node can be discarded. For example, if a partial schedule contains a hole on a resource (an interval of time over which it can be shown that no activity requiring the resource can execute), and an activity is scheduled after the hole for no good reason, then the node can be discarded since another branch will lead to a schedule in which this activity (or another) occupies the hole [48].

- An optimization criterion is called "sequence-dependent" if it depends only on the relative order in which activities are executed. Typical example are of course the sum of setup times and the sum of setup costs. When optimizing sequence-dependent criteria, it is once again particularly appropriate to solve the resource constraints by ordering activities: once activities are sequenced, the earliest start and end times that result from constraint propagation can be used as a solution. Note however that the dominance properties that exist for regular criteria cannot be applied to sequence-dependent criteria: for example, it might be worth leaving a hole in a schedule by executing a specific activity later if it enables to save a costly setup.

- Other optimization criteria are more difficult to optimize. For example, work in process time, *i.e.*, the average difference between the end time of the last activity composing a given job and the start time of the first activity of the job, is an irregular criterion which is difficult to optimize as the first activity of each job shall be executed as late as possible while the last activity of each job shall be executed as early as possible. Storage costs, in particular the cost of storing intermediate products, are difficult to optimize for the same reason. In such cases, it is not sufficient to sequence the activities. It is however often the case that once the resource constraints have been solved by sequencing activities, a linear program can be used to determine the optimal solution for the chosen sequences. Hybrid algorithms based on both CP and Mixed Integer Programming (MIP) [64] can be used for this purpose [2].

22.5.1 The Use of Local Search

Even when search can be simplified by looking for good sequences and using dominance properties, search spaces for planning or scheduling problems tend to be very large. In practice, it is often impossible to explore a search space completely and guarantee the delivery of an optimal solution. For an industrial planning or scheduling application it however generally suffices to provide "good" solutions within reasonable time. It is for such applications more important to be robust with respect to variations in the problem instances like variations in problem size, variations in numerical characteristics, and addition of side constraints. This is often achieved by mixing constraint-based tree search with Local Search (LS) or by actually implementing LS with constraints. Local search is taken as an alternative way to explore the search space. Explored neighborhoods vary a lot from an application to another, so it is difficult to establish a general taxonomy of the approaches reported in the literature. We will use two examples to convey the basic ideas.

Caseau and Laburthe [13] describe an algorithm for the Job Shop Scheduling Problem which combines CP and LS. The overall algorithm finds an approximate solution to start with, makes local changes and repairs on it to quickly decrease the makespan and, finally, performs an exhaustive search for decreasing makespans. Given a schedule, a critical path is defined as a sequence of activities where i) for each activity A_i that appears before activity A_j in the sequence A_i indeed precedes A_j in the schedule and ii) the sum of the processing times of the activities in the sequence equals the makespan of the schedule. Two types of local moves are considered:

- "Repair" moves swap two activities scheduled on the same machine to shrink or reduce the number of critical paths.
- "Shuffle" moves [1] keep part of the solution and search through the rest of the solution space to complete it. Each shuffle move is implemented as a constraint-based search algorithm with a limited number of backtracks (typically 10, progressively increased to 100 or 1000), under the constraint that the makespan of the solution must be improved (with a given improvement step, typically 1% of the makespan, progressively decreased to one time unit).

Excellent computational results have been obtained with this approach [13, 14] as well as with other constraint-based implementations of shuffle moves, as reported in [4, 55].

In the same spirit, the best algorithm used by Le Pape and Baptiste [46] for the Preemptive Job Shop Scheduling Problem relies on the combination of:

- a strong constraint propagation algorithm (edge-finding);
- a local optimization operator called "Jackson derivation";
- limited discrepancy search [38] around the best schedule found so far.

Limited discrepancy search is an alternative to depth-first search, which relies on the assumption that a heuristic makes few mistakes throughout the search. Thus, considering the path from the root node of the tree to the first solution found by a depth-first search algorithm, there should be few "wrong turns" (*i.e.*, few nodes which were not immediately selected by the heuristic). The basic idea is to restrict the search to paths that do not diverge more than w times from the choices recommended by the heuristic. Each time this limited search fails to improve on the best current schedule, w is incremented and the process is iterated, until either a better solution is found or it is proven that there is no better solution. It is easy to prove that when w gets large enough, limited discrepancy search is complete. Yet it can be seen as a form of LS around the recommendation of the heuristic. On ten well-known problem instances, each with 100 activities, experimental results show that each of the three techniques mentioned above brings improvements in efficiency, the average deviation to optimal solutions after 10 minutes of CPU time falling from 13.72% when none of these techniques is used to 0.23% when they are all employed.

Globally, the integration of LS and CP is promising whenever LS operators provide a good basis for the exploration of the search space and either side constraints or effective constraint propagation algorithms can be used to prune the search space. The examples presented in the literature represent a significant step toward the understanding of the possible combinations of LS and CP. Yet the definition of a general approach and methodology for integrating LS and CP remains an important area of research.

22.5.2 The Use of Mixed Integer Programming

In industrial applications, scheduling issues are often mixed with resource allocation, capacity planning, or inventory management issues for which MIP is a method of choice.

Several examples have been reported where a hybrid combination of CP and MIP was shown to be more efficient than pure CP or MIP models (cf., for example, [60, 61, 63, 2, 19, 43]). As in the case of local search, there are many ways to combine CP and MIP, and we will just focus on two particular examples.

A dynamic scheduling problem is solved in [63]. In this example, the linear solver includes only temporal constraints (some of which have been added to the initial problem in order to ensure the satisfaction of resource constraints) and the definition of the optimization criterion as the total deviation of start times of activities from the start times of the same activities in a reference schedule. An interesting characteristic of this model is that the optimal continuous solution of the linear sub-problem is guaranteed to be integral; hence, either this solution satisfies all the resource constraints and it is optimal, or it violates some resource constraint which can be used to branch on the order of two conflicting activities. CP is used to limit and select the explored branches.

[19] and [43] consider the case in which it is not certain that an activity will use a given resource, either because there are alternative resources, or because the activity can be left unperformed against a certain cost (see Section 22.1.4). We recall that an unperformed activity will not require capacity, but will obey potential temporal constraints, etc., and will also obey the calendar of the chosen resource. [19, 43] do not consider resource calendars, so an unperformed activity requires its normal processing time to be completed. Note that to our knowledge no MIP approach exists that handles resource calendars.

Also without resource calendars this problem is already challenging for any optimization technique. MIP is a good candidate for representing the cost function, but no good MIP model is known to state that a resource can only perform one activity at a time. CP usually deals well with precedence and resource constraints, but adding an upper bound on the optimization criterion does in general not result in effective constraint propagation. LS operators based on permuting activities are easy to design, but the impact of a permutation on the total cost is hard to estimate. In [19], several cooperative optimization algorithms centered on a MIP model have been proposed and compared with a pre-existing combination of CP and LS:

- The MIP algorithm relies on the default search strategy of CPLEX 9.0 [39].

- The IS+MIP algorithm consists in using CP to construct an initial solution to the problem. This solution is then used as a starting point for CPLEX.

- The IS+MIP+RINS algorithm is similar to IS+MIP but activates the relaxation induced neighborhood search option of CPLEX [20]. Relaxation induced neighborhood search is a form of LS which relies on the continuous relaxation to define a neighborhood of the current solution: the integer variables that have the same values in the solution of the continuous relaxation and in the best solution known so far are fixed to these values and a sub-MIP on the remaining variables is solved (with a limit on the number of nodes explored).

- The IS+MIP+RINS+GD algorithm adds the guided dives option of CPLEX [20] to the IS+MIP+RINS algorithm. When a variable is selected for branching, the "guided dives" strategy will explore first the node in which this variable is fixed to the value that it takes in the best solution known so far.

- The IS+MIP+RINS+GD+MCORE algorithm adds to the IS+MIP+RINS+GD algorithm another form of LS which defines a neighborhood by heuristically reducing the values of "big-M" coefficients of the MIP model.

These algorithms have been tested on 22 job shop instances from the Manufacturing Scheduling Library (MaScLib) [58], with up to 260 activities. The results have shown the interest of all the components that have been added to the initial MIP algorithm. They also show that on pure problems, hybrid algorithms based on MIP can compete with state-of-the-art techniques.

The generalization of these examples into a principled approach is an important research issue for the forthcoming years. In particular, MIP models are often difficult to extend to the representation of additional constraints such as setup times and costs, calendars, etc.

22.6 Conclusions

In the introduction of this chapter we have seen that one of the key factors of the success of Constraint-Based Scheduling lies in the fact that a powerful combination was found of the research fields of Operations Research (OR) and Artificial Intelligence (AI). From OR its efficient algorithms and its culture for searching for efficient algorithms were used. From AI the general modeling and problem-solving paradigm of CP and its culture for searching for natural ways of modeling a problem in the needed real-life detail were used.

In this way Constraint-Based Scheduling preserves the general modeling and problem-solving paradigm of CP while the integration of efficient propagation algorithms improves the overall performance of the approach. Efficient OR algorithms integrated in a CP approach allow the user to benefit from the efficiency of OR techniques in a flexible framework. Although the use of CP in planning is, due to the problem complexity, less mature than its use in scheduling, Constraint-Based Planning follows the same pattern as Constraint-Based Scheduling where CP is used as a framework for integrating efficient special purpose algorithms into a flexible and expressive paradigm. As in several other areas of application, an important way to integrate efficient algorithms in CP for scheduling and planning was found by incorporating them inside global constraints. Sections 22.3 and 22.4 pay attention to such constraint propagation.

Besides the powerful propagation, another strength was identified namely the capacity to in a natural and flexible way model the scheduling or planning problem at hand in the required real-life detail. We want to stress that this capacity is becoming more and more important. Indeed through the widespread adoption of ERP (Enterprise Resource Planning) systems, more and more companies have access to the data that allows them to capture the reality in the detail they need. One of the reasons Advanced Planning and Scheduling systems (APS's) are not as widely adopted as one would think following this observation, is that these offerings often fail to model reality in sufficient detail. This leads to the aforementioned classical drawbacks of one being forced to discard degrees of freedom and side constraints. It's especially on the side constraints that APS's tend to be weak, thus leading to the system solving an oversimplified problem resulting in producing impractical solutions for the original problem. It is here that we believe Constraint-Based Planning and Scheduling have a great, largely unused, potential.

Two other strengths identified in this chapter are i) a natural fit of expressing scheduling specific heuristics using CP tree search, and ii) a proven good potential of combining the CP approach with solution techniques as Local Search, Large Neighborhood Search, and Mixed Integer Programming. We have seen several examples of this in Section 22.5. These strengths are thus about having the flexibility in the approach to adapt the search such that the needed performance to solve the problems is obtained. Although this has indeed been a strength over the years, we want to stress that we believe the field should pay increased attention to providing good default search, *i.e.*, a search procedure that works "out-of-the-box" at least for a certain class of problems. This is much like a lot of the work done in the area of Mixed Integer Programming. That latter work has led to a broadening of the audience that can use Mixed Integer Programming to solve their problems. A similar effect should be obtained for CP in general and Constraint-Based Planning and Scheduling in particular. This, combined with the natural way of modeling problems present in CP, should open up CP for a much broader use than today.

Another main research challenge is on doing planning and scheduling under uncertainty. Uncertainty is inherent to planning and scheduling environments and correctly dealing with it is of invaluable practical importance. Two basic ways for dealing with uncertainty, together with different combination of them, have been studied: reactive (rescheduling) and proactive (robust scheduling). Lots of research has been done starting many years ago but surprisingly few approaches have been applied in practice. We feel the field is ripe to adopt rescheduling and robust scheduling more broadly and believe CP can play an important role there.

Further, more detailed, research challenges link back to the strengths already mentioned. It remains a challenge to study industrial properties in detail. Studies around breakable activities, productivity profiles, continuous production and consumption, unperformed activities, etc., are rare while there is a great need in practice to correctly handle such properties.

In Section 22.4 constraint propagation methods related to the minimization of the weighted number of late activities and to the minimization of setup times and setup costs have been presented. They drastically improve the behavior on problems involving these criteria. However, there are many other interesting optimization criteria. In particular, total tardiness is widely used in industry but until now poor results are obtained on this problem. Constraint propagation on such specific optimization criteria constitutes a very interesting research area. Following the observation that users of planning and scheduling applications often want to define their own criteria, a possibly even more interesting research challenge is to design generic lower-bounding techniques and constraint propagation algorithms that could work for any criterion.

Finally, a research challenge in Constraint-Based Planning is to still better exploit the combination of AI and OR, *i.e.*, to continue to follow the same pattern as Constraint-Based Scheduling where CP is used as a framework for integrating efficient special purpose algorithms into a flexible and expressive paradigm. This will bring all the strengths of Constraint-Based Scheduling mentioned in this chapter to Constraint-Based Planning.

Bibliography

[1] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

[2] Ph. Baptiste and S. Demassey. Tight LP bounds for resource constrained project scheduling. *OR Spektrum*, 26:251–262, 2004.

[3] Ph. Baptiste and C. Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proc. 15th Workshop of the U.K. Planning Special Interest Group*, 1996.

[4] Ph. Baptiste, C. Le Pape, and W. Nuijten. Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proc. 1st International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

[5] Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, 2001.

[6] A. Blum and M. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[7] P. Brucker and O. Thiele. A branch and bound method for the general shop problem with sequence dependent setup-times. *OR Spektrum*, 18:145–161, 1996.

[8] J. Carlier. *Problèmes d'Ordonnancement à Contraintes de Ressources : Algorithmes et Complexité*. Thèse de doctorat d'Etat, Université Paris VI, 1984. In French.

[9] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[10] J. Carlier and E. Pinson. A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[11] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[12] Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In *Proc. 11th International Conference on Logic Programming*, 1994.

[13] Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical report, Ecole Normale Superieure, 1995.

[14] Y. Caseau, F. Laburthe, C. Le Pape, and B. Rottembourg. Combining local and global search in a constraint programming environment. *Knowledge Engineering Review*, 16:41–68, 2001.

[15] A. Cesta and A. Oddi. Gaining efficiency and flexibility in the simple temporal problem. In *Third International Conference on Temporal Representation and Reasoning (TIME-96)*, 1996.

[16] A. Cesta and C. Stella. A time and resource problem for planning architectures. In *ECP-97*, 1997.

[17] D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 33:333–377, 1987.

[18] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling. In *Proc. SpaceOps 2000*, 2000.

[19] E. Danna. *Intégration des techniques de recherche locale à la programmation linéaire en nombres entiers*. PhD thesis, Université d'Avignon et des Pays de Vaucluse, 2004.

In French.

[20] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.

[21] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–96, 1991.

[22] M. Dell'Amico and S. Martello. Linear assignment. In *Annotated Bibliographies in Combinatorial Optimization*. John Wiley and Sons, 1997.

[23] M.B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.

[24] J. Erschler. *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement*. Thèse de doctorat d'etat, Université Paul Sabatier, 1976.

[25] J. Erschler, P. Lopez, and C. Thuriot. Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnacement. *Revue d'Intelligence Artificielle*, 5:7–32, 1991. In French.

[26] R. Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[27] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[28] F. Focacci. *Solving Combinatorial Optimization Problems in Constraint Programming*. PhD thesis, Università di Ferrara, 2001.

[29] F. Focacci and W. Nuijten. A constraint propagation algorithm for scheduling with sequence dependent setup times. In *Proc. CPAIOR '00*, 2000.

[30] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proc. Fifth International Conference on Artificial Intelligence Planning and Scheduling, AIPS'00*, pages 92–101. AAAI Press, 2000.

[31] M.S. Fox. Constraint-guided scheduling: A short history. *Computers in Industry*, 14: 79–88, 1990.

[32] J. Frank and A. Jónsson. Constraint-Based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003.

[33] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Wiley & Sons, 1982.

[34] S. Martello G. Carpaneto and P. Toth. Algorithms and code for the assignment problem. *Annals of Operations Research*, 13:193–223, 1988.

[35] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proc. AIPS-94*, pages 61–67, 1994.

[36] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[37] M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley and Sons, 1984.

[38] W. Harvey and M. Ginsberg. Limited discrepancy search. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1995.

[39] ILOG CPLEX. *ILOG CPLEX 9.0 User's Manual and Reference Manual*. ILOG, S.A., Gentilly, France, 2003.

[40] A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *Proc. AIPS-00*, 2000.

[41] P. Laborie. Algorithms for propagation resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143:151–188, 2003.

[42] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proc. IJCAI-95*, pages 1643–1649, 1995.

[43] C. Le Pape. Experiments with cooperative optimization algorithms for production scheduling. In *Proc. Oberwolfach Workshop on Mathematics in the Supply Chain*, 2004.

[44] C. Le Pape. *Des systèmes d'ordonnancement flexibles et opportunistes*. PhD thesis, University Paris XI, 1988. In French.

[45] C. Le Pape. Implementation of resource constraints in ILOG Schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.

[46] C. Le Pape and Ph. Baptiste. Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics*, 5:305–325, 1999.

[47] C. Le Pape and S. F. Smith. Management of Temporal Constraints for Factory Scheduling. In F. Bodart C. Roland and M. Léonard, editors, *Temporal Aspects in Information Systems*. North-Holland, 1988.

[48] C. Le Pape, P. Couronné, D. Vergamini, and V. Gosselin. Time-versus-capacity compromises in project scheduling. *AISB Quarterly*, 91:19–31, 1995.

[49] O. Lhomme. Consistency techniques for numeric CSPs. In *Proc. 13th International Joint Conference on Artificial Intelligence*, 1993.

[50] H. C. R. Lock. An implementation of the cumulative constraint. Working Paper, University of Karlsruhe, 1996.

[51] A. Lopez and F. Bacchus. Generalizing GraphPlan by Formulating Planning as a CSP. In *Proc. IJCAI-03*, pages 954–960, 2004.

[52] P.D. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job shop scheduling problem. In S.T. McCormick W.H. Curnigham and M. Queyranne, editors, *Proc. Fifth International IPCO conference, Vancouver, Canada*, pages 389–403. LNCS 1084, 1996.

[53] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proc. AAAI-91*, pages 634–639, 1991.

[54] W. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, 1994.

[55] W. Nuijten and C. Le Pape. Constraint-based job shop scheduling with ILOG Scheduler. *Journal of Heuristics*, 3:271–286, 1998.

[56] W.P.M. Nuijten, E.H.L. Aarts, D.A.A. van Erp Taalman Kip, and K.M. van Hee. Job shop scheduling by constraint satisfaction. Computing Science Note 93/39, Eindhoven University of Technology, 1993.

[57] W. Nuijten and F. Sourd. New time-bound adjustment techniques for shop scheduling. In *Proc. PMS 2000*, pages 224–226, 2000.

[58] W. Nuijten, T. Bousonville, F. Focacci, D. Godard, and C. Le Pape. Towards an industrial manufacturing scheduling problem and test bed. In *Proc. 9th International Conference on Project Management and Scheduling*, pages 162–165, 2004.

[59] L. Peridy. *Le problème de job-shop : arbitrage et ajustements*. PhD thesis, Université de Technologie de Compiègne, 1996. In French.

[60] R. Rodosek and M. Wallace. A generic model and hybrid algorithm for hoist scheduling problems. In *Proc. 4th International Conference on Principles and Practice of Constraint Programming*, 1998.

[61] R. Rodosek, M. Wallace, and M. T. Hajian. A new approach to integrating mixed inte-

ger programming and constraint logic programming. *Annals of Operations Research*, 86:63–87, 1999.

[62] E.D. Sacerdoti. A Structure for Plans and Behaviours. Technical Note 109, SRI, 1975.

[63] H. El Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5:359–388, 2000.

[64] A. Schrijver. *Combinatorial Optimization*. Springer-Verlag, 2003.

[65] C. Schwindt. *Resource Allocation in Project Management*. Springer-Verlag, 2005.

[66] D.E. Smith, J. Frank, and A.K. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[67] S. F. Smith. OPIS: A Methodology and Architecture for Reactive Scheduling. *Intelligent Scheduling*, 1994. M. Zweben and M. Fox (editors). Morgan Kaufmann.

[68] S.F. Smith and C.-C. Cheng. Slack-based heuristics for constraint satisfaction. In *Proc. 11th National Conference on Artificial Intelligence*, 1993.

[69] F. Sourd and W. Nuijten. Multiple-machine lower bounds for shop scheduling problems. *INFORMS Journal on Computing*, 12(4):341–352, 2000.

[70] G.L. Steele, Jr. *The Definition and Implementation of a Computer Programming Language Based on Constraints*. PhD thesis, Massachusetts Institute of Technology, 1980.

[71] Ph. Torres and P. Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127:332–343, 2000.

[72] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[73] P. van Beek and X. Chen. A constraint programming approach to planning. In *Proc. AAAI-99*, pages 585–590, 1999.

[74] C. Varnier, P. Baptiste, and B. Legeard. Le traitement des contraintes disjonctives dans un problème d'ordonnancement : exemple du hoist scheduling problem. In *Actes 2èmes journées francophones de programmation logique*, 1993.

[75] V. Vidal and H. Geffner. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. *Artificial Intelligence*, 2005. To appear.

[76] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of Fifth National Conference on Artificial Intelligence*, pages 377–382, 1986.

[77] P. Vilim. O(n log n) filtering algorithms for unary resource constraint. In *Proc. CPAIOR '04*, pages 319–334, 2004.

[78] P. Vilim, R. Bartak, and O. Cepek. Unary resource constraint with optional activities. In *Proc. CP 2004*, pages 62–76, 2004.