Scheduling Constraints

(Seminar: Constraint Programming)

Patrick Pekczynski
Tutor: **Dipl.-Inf. Guido Tack**

Department of Computer Science
Programming Systems Lab
Saarland University, 66041 Saarbrücken, Germany,
`pekczynski@ps.uni-sb.de`,

Abstract. The essence of this paper is an introduction into the *theory of scheduling* and the presentation of *constraint-based scheduling* as a viable solution method for the combinatorial hard scheduling problems occuring in this research area. The paper covers a definition and a *standard classification* for scheduling problems, their encoding into a *constraint satisfaction problem* as well as the definitions and propagation rules for specialised global *scheduling constraints* lying at the heart of *constraint-based scheduling.*

1 Introduction

The framework of *constraint programming (CP)* provides a viable approach for modeling and solving *hard combinatorial problems.* In this context, the notion of a *combinatorial problem* denotes "the problem of finding an object with some desired property among a finite set of possible alternatives"[vH05]. Various *combinatorial problems* occur in different real-life applications of *scheduling* and are called *scheduling problems.* In fact, it is not straightforward to solve these because the majority of them turns out to be $\mathcal{NP}$-hard [Sha05a,LdB04], i.e. there exists no polynomial time algorithm in the size of the problem specification to solve the problem. Nevertheless, *constraint-based scheduling (CBS)* provides an efficient CP-based method to solve scheduling problems. In order to explain the basics of CBS, the paper is split up in three parts: the first part of the paper focuses on scheduling in general, where sections 1.1 and 1.2 provide an overview and a definition of *scheduling.* Subsequently the sections 1.3 and 1.4 describe the basic concept of an activity in scheduling and introduce the standard $\alpha|\beta|\gamma$-classification for scheduling problems. The second part of the paper covers essential parts of CBS, where section 2.1 describes the basic components of CBS. The last two sections in this part, section 2.2 and section 2.3 explain the encoding of a scheduling problem as a *constraint satisfaction problem (CSP)* and furnish definitions as well as formal propagation rules for the global scheduling constraints lying at the heart of CBS. Finally, the paper's last part focuses on specialised branching strategies for CBS in contrast to standard branching schemes commonly used in CP.

1.1 Scheduling - A brief overview

To get a clear idea of what CBS is dealing with, this section provides a brief overview of scheduling theory and application. Scheduling theory first appeared in the early 1950s and is embedded in three different research areas. Since *Operations Research (OR)* is one of these, scheduling problems can be found in many real-life applications like timetabling at university, determining an efficient execution order of processes in operating - or computer systems, staff assignment in hospitals, vehicle routing in transport or developing efficient production schedules in manufacturing. As a subfield of *Mathematical Programming (MP)*, scheduling problems imply the problem of *minimising or maximising a function* ranging over variables being constrained to fulfill predefined conditions. In addition to that, scheduling is also a subfield of *combinatorial optimisation (CO)*. Hence, scheduling problems not only deal with an efficient allocation of limited resources, but also try to find the best optimal solution(s) from a *finite* set of feasible solutions. These relations to other research areas provide a good intuition of scheduling as an efficient allocation of activities to resources, that has to be optimised with respect to an objective function. In order to clarify, what scheduling actually does, *figure 1* gives a schematic trace of the use of scheduling in the context of an advanced planning and scheduling system (APSS). Initially, there is a *user* desiring a product like a staff assignment, a plan for a pizza delivery or a timetable for lectures at a university. The *planner* determines a sequence of actions, a *plan p*, satisfying this custom order and producing the desired product, i.e. planning maps products to plans. The developed plan p is passed on to a *scheduler*, who decides at what time which resources are used to perform the activities of p, i.e. scheduling maps activities and their respective operating times to resources. After having transfered the determined points in time for execution to the *executor*, the *system* is finally able to send the resulting product back to the user.

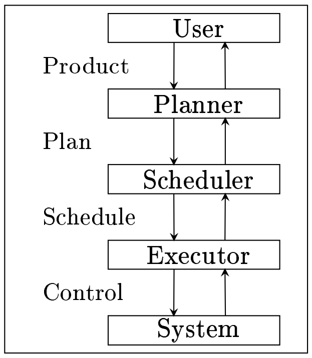

Fig. 1. APSS-scheme [Fro01]

1.2 Definition of Planning and Scheduling

From the context of this traditional APSS-scheme we learn, that the only application areas for CP are the *planning-* and *scheduling*-phases.

Definition 1 (Planning). *Planning is the process of finding a sequence p of actions a_i, such that an initial state s_0 is transfered into a goal state s_g (the custom order) [Bar04]. The resulting sequence $p:=< s_0, a_0, \ldots, a_n, s_g >$ is called a* plan. *Thus, planning is defined as a mapping $\mathcal{P}$ from a set $\mathcal{P}r$ of products* *(custom orders or desired world states) to a set $\mathcal{Pl}$ of plans as activity lists producing the desired states ($\mathcal{P}$: $\mathcal{Pr} \mapsto \mathcal{Pl}$).*

In [Bar04], planning is considered a *dynamic problem*, because the activities a_i transfering s_0 into s_g are unknown in advance. However, it is possible to reduce the dynamic planning problem to a static one by assuming, that only plans of a certain length n are taken into account [Bar04] providing static information about the activities. Hence, it becomes possible to model such a planning problem as a *constraint satisfaction problem (CSP)*. In addition to the embedding in other research areas, the application of scheduling in an APSS (cf. *figure 1*) underlines, that scheduling is a mapping from activities to resources. Together with the brief characterization of scheduling as a special branch in OR, MP and CO, this observation is congruent with the following definition of scheduling, given in the relevant technical literature.

Definition 2 (Scheduling). *Scheduling is the allocation of scarce resources over time. [Bak74] Let $\mathcal{Act}$ denote the set of available activities, that have to be performed, $\mathcal{Res}$ denote the set of available limited resources on which the activities are executed, and $\mathcal{T}$ a set of points in time describing when the activities start their execution on the respective resources. Then, a schedule $\mathcal{S}$: $\mathcal{Act} \times \mathcal{T} \mapsto \mathcal{Res}$ maps activities $a_i \in \mathcal{Act}$ together with their respective starting times $\in \mathcal{T}$ to limited available resources $R_j \in \mathcal{Res}$. Thus, scheduling determines exactly, what activities are performed on which resource and at which time.*

In contrast to planning, scheduling is a *static problem* [Bar04], because the activities a_i subject to a schedule $\mathcal{S}$ are known in advance, i.e. informations about their start, processing and end times are already provided. Therefore it is possible to model each scheduling problem as a *constraint satisfaction problem (CSP)*. Although CP is applicable in both areas, the following sections only focus on the scheduling component as background for CBS and the scheduling constraints.

1.3 From Activities to Jobs

According to *definition 2*, scheduling is a mapping from a set of activities to a set of resources.

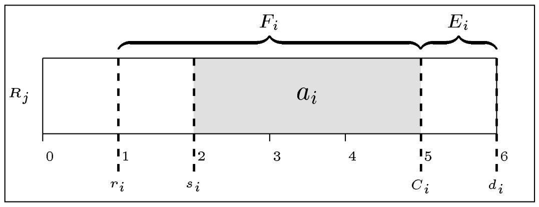

Fig. 2. Early Activity [BLPN01]

This implies that both concepts, activities and resources, represent the fundamental components of the CSP-encoding for a scheduling problem. Since the characteristics of an activity are not only an essential part of the CSP-model, but also of the standard classification for scheduling problems (cf. *section 1.4*), they are explained in detail in this section. Consider an activity a_i, which has to be scheduled on a resource R_j. Such an a_i is determined by the following quantities shown in *figure 2* and *figure 3*: r_i specifies the *release date* of the activity, i.e. the time, at which, a_i can start, d_i specifies the *deadline* of the activity, i.e. the time, before or at which a_i must have ended, s_i specifies the *start time* of the activity, i.e. time, at which a_i actually starts, and $e_i (= C_i)$ specifies the *end time (completion time)* of the activity, i.e. the time, at which a_i actually ends. $F_i = C_i - r_i$ specifies the *flow time*, i.e. how long the activity stays ready in the system until it finishes execution. $L_i = C_i - d_i$ describes the *lateness*, i.e. in case of violation of the activity's deadline the lateness denotes the time from deadline to completion time of the activity, whereas $-L_i$ denotes the difference between deadline and completion time if a_i finishes before its deadline. $D_i = max\{0, L_i\}$ specifies the *tardiness* of an activity (cf. *figure 3*), i.e. whether an activity is late ($D_i = L_i$) or not ($D_i = 0$) and $E_i = max\{0, -L_i\}$ specifies the *earliness* of an activity (cf. *figure 2*), i.e. whether an activity is early ($E_i = -L_i$) or not ($E_i = 0$).

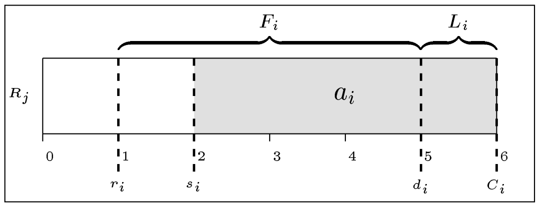

Fig. 3. Late Activity [BLPN01]

The concept of activities, that is used throughout scheduling theory can be extended to *jobs*. If we consider activities $\{a_1, \ldots, a_n\}$ as depicted in *figure 2* or *figure 3*, a job J_m is defined as the rearrangement of these activities into a strongly related subset $J_m = \{a_1, \ldots, a_k\}$ [BLPN01]. If $n = 1$ the terms job and activity are used synonymously. Furthermore, the activities in J_m are linked by a set of *precedence constraints*, whereof the most common form is a *chain-like* routing on the activities,

$$a_1 \rightarrow \ldots \rightarrow a_k$$

Finally, as a job consists of a set of activities, each job can be determined by release date, deadline, start and end times.

1.4 $\alpha|\beta|\gamma$-Classification of Scheduling

As stated in [Sha05b], scheduling theory covers more than 10.000 different problem types and is characterised by a "virtually unlimited number" [Bru98] of them. Because of the lack of a uniform description for scheduling problems, in 1979 *Graham et al.* introduced a formal classification system to describe scheduling problems. With respect to further extensions, the $\alpha|\beta|\gamma$-classification they introduced is widely used as standard classification for scheduling problems [BLPN01,Sha05b,LdB04,Kri04,Leg04] and bases on a *three-field-descriptor* $\alpha|\beta|\gamma$. This section describes the original classification without extensions. If $\circ$ denotes the absence of a symbol, we obtain the following properties for the α, β and γ parameters:

Resource Environment α

Instead of **resource environment**, in the literature the α-descriptor is often referred to as *machine enviroment* or *processor environment.* The first field

$$\alpha := \alpha_1,\ \alpha_2$$

consists of two further subfields and denotes the *machine environment* of a scheduling problem. In this context, $\alpha_1 \in \{\circ, P, Q, R, F, O, J\}$ specifies the machine type (cf. *table 1*) and α_2 denotes the number of machines in the problem, which varies between $\circ$ for an arbitrary number of machines and $k \geq 1$ for a fixed number k of machines.

$\circ$	= single resource
P	= parallel resources
Q	= uniform resources
R	= unrelated resources
F	= Flow-Shop system
O	= Open-Shop system
J	= Job-Shop system

Table 1. Values for α_1

If $\alpha_1 \in \{F, O, J\}$ all machines are dedicated, i.e. each machine services one primary function or task [Kri04]. If $\alpha_1 = F$, then each job J_l follows the same routing, i.e. each job J_l executes its activities $a_{l,i}$ on a fixed machine order $\mathcal{F} =< m_a, \ldots, m_b >$. In case, that $\alpha_1 = J$, each job J_l has its on machine routing. If $\alpha_1 = O$, the machine routing can be arbitrary [Sha05b].

Activity and Resource Characteristics β

The second field

$$\beta := \beta_1,\ \beta_2,\ \beta_3,\ \beta_4,\ \beta_5,\ \beta_6,\ \beta_7,\ \beta_8$$

has 8 sub-categories and represents the **activity and resource characteristics**, which is also referred to as *job characteristics* [BLPN01] or *task and resource environment*[Kri04].

$\circ$	= independent activities
prec	= directed acyclic graph
uan	= uniconnected
	= activity networks
tree	= tree structure
chains	= set of chains

Table 2. Values for β_3

The first subfield β_1 indicates, whether activities are *preemptive* ($\beta_1 = pmtn$) or not ($\beta_1 = \circ$). In this case, the property of preemption means, that the execution of an activity can be interrupted, e.g. by the execution of an other activity on the same resource. β_2 states, whether there are additional resource constraints ($\beta_2 = res\lambda,\ \delta,\ \rho$) or not ($\beta_2 = \circ$). According to [?] $\lambda,\ \delta,\ rho$ denote the number of resource types(λ), resource limits(δ) and the resource requirements(ρ). These values either can be arbitrary ($\lambda,\ \delta,\ \rho = \circ$) or fixed to some $k \geq 1$, stating that there are k resources types in the system ($\lambda = k$), each resource in the system has a maximal capacity of$k$ ($\delta = k$) and each activity demands $c \leq k$ units of a resource ($\rho = k$). $\beta_3 \in \{\circ, prec, uan, tree, chains\}$ illustrates the kind of precedence between tasks and indicates the structure of the precedence graph (cf. *table 2*), if there is any precedence relation on the activities in focus. β_4 shows, whether the ready times

of all activities are zero ($\beta_4 = \circ$) or whether there are different ready times for an activity a_j ($\beta_4 = r_j$). The fifth subfield indicates, if the activities have arbitrary processing times ($\beta_5 = \circ$), if all activities have the same processing time p ($\beta_5 = (p_j = p)$) or if every processing time is bounded by the interval $[\underline{p}, \overline{p}]$ ($\beta_5 = \underline{p} \leq p_j \leq \overline{p}$).
Parameter β_6 describes whether there are no deadlines imposed on the activities ($\beta_6 = \circ$) or whether deadlines are imposed on the performance of some activities ($\beta_6 = \tilde{d}$). In case of a job-shop system ($\alpha_1 = J$), β_7 denotes the maximal number of activities forming a job, which can be either arbitrary ($\beta_7 = \circ$) or bounded by some fixed k for each job ($\beta_7 = n_j \leq k$). The last parameter β_8 denotes the *no-wait*-property, i.e. indicates, whether buffers of unlimited capacity are assumed ($\beta_8 = \circ$) or whether buffers between machines have zero capacity, that is, a job finishing on one machine must immediately start on the next machine.

$\frac{1}{n} \cdot \sum_{j=1}^{n} F_j$	= mean flow time $\overline{F}$
$\frac{\sum_{j=1}^{n} w_j \cdot F_j}{\sum_{j=1}^{n} w_j}$	= mean weighted flow time $\overline{F}_w$
$max\{L_j\}$	= maximum lateness L_{max}
$\frac{1}{n} \sum_{j=1}^{n} D_j$	= mean tardiness $\overline{D}$
$\frac{\sum_{j=1}^{n} w_j \cdot D_j}{\sum_{j=1}^{n} w_j}$	= mean weighted tardiness $\overline{D}_w$
$\frac{1}{n} \sum_{j=1}^{n} E_j$	= mean earliness $\overline{E}$
$\frac{\sum_{j=1}^{n} w_j \cdot E_j}{\sum_{j=1}^{n} w_j}$	= mean weighted earliness $\overline{E}_w$
U_j	$= \begin{cases} 1, \text{if } C_j > d_j \\ 0, \quad \text{otherwise} \end{cases}$
$\sum_{j=1}^{n} U_j$	= # tardy tasks U
$\sum_{j=1}^{n} w_j \cdot U_j$	= weighted # of tardy tasks $\overline{U}_w$
$-$	= testing for feasibility

Table 3. Values for γ

Optimality Criterion γ

The γ-field represents the optimality criterion for the scheduling problem. Thus, $\gamma \in \{C_{max}, \overline{F}, \overline{F}_w, L_{max}, \overline{D}, \overline{D}_w, \overline{E}, \overline{E}_w, U, \overline{U}_w, -\}$ is a measure for the quality and the performance of the schedule and contains the objective function of problem structure, which has to be optimised. The most common optimisation criteria are the maximum completion time,

$$C_{max} = max_{j \in \{1,\ldots,n\}}\{C_j\}$$

(cf. *section 1.3*), which equals the maximum schedule length or which is also referred to as maximum *makespan*, the total completion time,

$$\sum_{j=1}^{n} C_j$$

or the weighted total completion time,

$$\sum_{j=1}^{n} w_j \cdot C_j$$

An overview of other optimality criteria occurring as values for γ can be found in *table 3*.
The systematic notation of the complete $\alpha|\beta|\gamma$-classification as stated above serves as standard classification for scheduling. This classification scheme not only generalises the problem structure of scheduling problems, but for the first time allows a discussion of these problems by offering a uniform notation handling different scheduling problems. Further examples can be found in [cla,ORc]
With this standard notation and the basic knowledge and specifications about scheduling from the *sections 1.1 to 1.4* at hand, the next step is to describe the components of CBS.

2 Constraint-Based Scheduling

In this part of the paper we concentrate on the integral components of CBS like the CSP-model for scheduling problems and specialised scheduling constraints, and we trace the origins of CBS. Therefore, the following section takes a closer look at the development of CBS.

2.1 Components of CBS

CBS originates from four different research areas. One of these four components of CBS is OR, which is outlined in **section 1.1**. In order to solve scheduling problems, the OR-approach focuses exclusively on simple mathematical and statistical models by heavily exploiting the combinatorial problem structure [BLPN01]. Using these models the OR-method often has to discard side constraints and degrees of freedom. Although this modus operandi does not loose solutions, it works on a simpler form of the original problem. Thus a solution to this simpler problem may either be just a local optimum and hence not the optimal solution of the orginal problem or is not practicable in real-life applications of the problem. Notwithstanding, this mathematical approach yields efficient algorithms, which are used in specialised global constraints presented in *section 2.3*. In contrast to OR, *artificial intelligence (AI)*, another component of CBS, concentrates on more general models for scheduling problems without discarding any side-constraints. Thus, the AI-approach yiels algorithms that apply on more scheduling problems because, but as AI takes all side-constraints of the problem into consideration, the AI-algorithms only come up with a poor performance in comparison to the faster OR-algorithms. In spite of the computational power of OR and the general model of AI, neither OR, nor AI as “stand-alone-approaches” are able to handle scheduling problems, because OR-algorithms may perform only local optimisation and AI-algorithms are too slow. Together with CP as a third component of CBS however, OR and AI are finally able to tackle the problems occuring in scheduling theory, which forms the fourth and last component of CBS. Thus, the possibility of CP to compile scheduling problems into a CSP enables CBS to handle the $\mathcal{NP}$-hard scheduling problems. In addition to that, the applied constraint solving techniques of constraint propagation and constraint distribution are enhanced with OR-algorithms forming specialised global scheduling constraints and with-AI heuristics for the distribution part.

2.2 CSP-Encoding [BLPN01]

In this section we will describe the encoding of a scheduling problem as a *constraint satisfaction problem (CSP).* As the classification mentionned in *section 1.4* confronts us with a rather large number of problem variations, we restrict the CSP-model in focus to a CSP-model for *non-preemptive* scheduling problems, i.e. problem formulations where $\beta_1 = \circ$ holds. In order to compile a scheduling problem into a CSP, we use the definition of a CSP as it is stated in [Apt03], where a CSP is defined as a tuple $<\mathcal{X}, \mathcal{D}, \mathcal{C}>$, with $\mathcal{X}$ as the set of variables, $\mathcal{D}$ as the set of their respective domains and $\mathcal{C}$ as the set of constraints over variables $x_i \in \mathcal{X}$. Hence, the resulting encoding looks as follows:

Variables $\mathcal{X}$ [BLPN01]

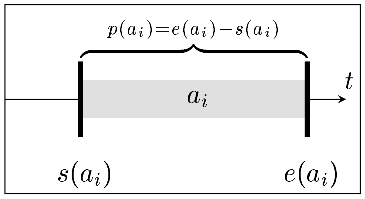

Fig. 4. Encoding an Activity [BLPN01]

As activities are the main concept of scheduling, the set of variables $\mathcal{X}$ denotes the available informations about the activities a_i of the scheduling problem. Taking only care of non-preemptive problems ($\beta_1 = \circ$) leads us to the following formulation: For each activity a_i in the problem definition, we introduce 3 variables $s(a_i), e(a_i)$ and $p(a_i)$ as is depicted in **figure 4**. The first variable, $s(a_i)$, denotes analogously to **section 1.3** the starting time of an activity a_i. Further, the second variable, $e(a_i)$, denotes the end of an activity a_i. The third an last variable $p(a_i)$ denotes the processing time of an activity a_i that is fixed in the problem description and defined as

$$p(a_i) = e(a_i) - s(a_i) \Leftrightarrow s(a_i) + p(a_i) = e(a_i)$$

As this equation holds, changing the domain of the $s(a_i)$-variable is coupled with changing the domain of the $e(a_i)$-variable and increases the propagation power of the posted constraints (cf. *section2.3*). Apart from the variables for the activities, an additional variable var_{of} is introduced that encodes the objective function in the problem structure and is used as optimality criterion during *branch and bound*-search (cf. *section 3*).

Domains $\mathcal{D}$ [BLPN01]
The domain modeling for the $\mathcal{X}$ -variables is essential for the constraints to follow and it reflects the restriction to $\beta_1 = \circ$. Since we find ourselves in the non-preemptive case, the variables in $\mathcal{X}$ range over a *time-window*

$$w = [r_i, d_i]$$

where r_i and d_i are defined as in *section1.3*. The definition of w indicates that an activity a_i is only allowed to execte between its release in the system and its specific deadline. Consequently, the variables in our model range over the following domains: The domain bounds for the respective variables defined above are initialised as follows: $D_i(s(a_i)) = [r_i, lst_i]$, $D_i(e(a_i)) = [eet_i, d_i]$ and $D_i(p(a_i)) = v_i$. v_i is a fixed known in advance value for the processing time of a_i, i.e. $p(a_i)$ is already determined by initialisation. lst_i denotes the *latest starting time* of an activity a_i and is defined as the maximal value in w that can be assigned to $s(a_i)$ without violating the deadline d_i. Similarly, eet_i denotes the *earliest end time* of an activity a_i and is defined as the minimal value in w that can be assigned to $e(a_i)$, i.e.

$$e(a_i) = eet_i \Leftrightarrow s(a_i) = r_i$$

The domain of var_{of} is initialised with the respective minimal and maximal value the objective function specified by the γ-descriptor can take with respect to the intial domain bounds of $s(a_i), e(a_i)$ and $p(a_i)$.

Constraints $\mathcal{C}$ [BLPN01]
The most important part of the CSP-tuple in CBS is the constraint set $\mathcal{C}$, incorporating the specialised global scheduling constraints. Essentially, the constraints of a scheduling problem can be split up into **temporal constraints**, **resource constraints** and the constraint, that the optimality criterion from the objective function is met. As it comes to temporal constraints, common constraints are precedence constraints like precedence between two activities $(e(a_i) \leq s(a_j))$ or precedence between two jobs $(e(j_i) \leq s(j_j))$. In addition to those precedence constraints, $\mathcal{C}$ also contains linear equations $s(a_i) + p(a_i) = e(a_i)$ which are due to non-preemption and increase propagation power since changing $D(s(a_i))$ implies changing $D(e(a_i))$ and vice versa. (cf. *Variables $\mathcal{X}$*). Apart from the temporal constraints the set $\mathcal{C}$ also contains resource constraints. Resource constraints model the fact, that an activity a_i requires k units of a resource R_j at some time t, which is denoted by $cap(a_i, t, R_j)$. If time t is not important, the resource usage of a_i is also denoted by $cap(a_i, R_j)$. Since this case is the most common for non-preemptive scheduling problems, we restrict the resource amount an activity uses to be $cap(a_i, R_j) = 1$. In the context of resource usage there is another important property, that has to be modeled, namely the capacity of a resource R_j described by $cap(R_j)$. The capacity of a resource indicates, how many activities a_i can be scheduled simultaneously on that resource. There is either the case, that only one activity can use the resource R_j at time t $(cap(R_j) = 1)$ or that the number of activities using resource R_j is bound

from above by $cap(R_j)$, i.e. $n \leq cap(R_j)$ activities can use R_j simultaneously. Obviously, the first case is a special case of the latter for $cap(R_j) = 1$. These two restrictions to the resource usage yield the most important constraints in CBS, the **unary resource constraint**

$$\forall t \in [0, T_{max}] : \sum_{a_i} cap(a_i, t, R_j) \leq 1$$

and the **cumulative constraint**

$$\forall t \in [0, T_{max}] : \sum_{a_i} cap(a_i, t, R_j) \leq cap(R_j)$$

stating that at each time t either **only one** activity (unary resource) or **at most** $cap(R_j)$ activities (cumulative) may use R_j, where t ranges over $[0, T_{max}] = [0, n \cdot max_{i\in\{1,\dots,n\}} p_i]$ according to the general concept of an activity (cf. *section 1.3*).

2.3 Scheduling Constraints

In this section we focus on *scheduling constraints* and on constraint propagation rules for them. The term scheduling constraints describes all constraints in $\mathcal{C}$ in the CSP-encoding of a scheduling problem. As we have seen, $\mathcal{C}$ includes two types of constraints, temporal and resource constraints. Temporal constraints express local properties. A precedence constraint denotes a local non-overlapping property $e(a_i) < s(a_j)$ and a linear constraint denotes local duration property $s(a_i) + p(a_i) = e(a_i)$. Therefore, temporal constraints can be propagated locally using linear equations for the duration property and inequalities for the non-overlapping property. The propagation of equations and inequalities is in $O(1)$ since for a temporal constraint, there are at least two and at most three variables involved. Without any precedence constraint, the linear constraints for all considered activities can be propagated in $O(n)$. Yet, taking precedence constraints into account, there can be at most $\binom{n}{2}$ of them resulting in a propagation complexity of $O\left(\binom{n}{2}\right) = O\left(\frac{n\cdot(n-1)}{2}\right) = O(n^2)$ for all temporal constraints. This means we have to post at most quadratically many local constraints to meet all temporal constraints. In contrast to the temporal constraints, resource constraints express the global property that for all time t the resource capacity is never exceeded [BLPN01]. For propagation of this property, four global propagation algorithms are proposed in the literature [Bar04,LN02,BLPN01]. They are based on the time-window notation w introduced in *section 2.2* and impose bounds consistency on the variable domains, i.e. prune the upper and lower bounds of the variable domains. Let $\Gamma = \{a_1, \dots, a_n\}$ denote a set of all available activities for a resource R_j with $|\Gamma| = n$ and lst_i,eet_i as defined as in *section 1.3*.

Definition 3 (timetable [BLPN01]).
timetable*(Γ) $\hat{=}$ lookup in an explicit global data-structure called* time-table*, whether at any time $t \in [0, T_{max}]$* only *$c = cap(R_j)$ activities $a_i \in \Gamma$ are scheduled on R_j:*

$$\begin{align} eet_i > lst_i &\Rightarrow \forall t \in [lst_i, eet_i] : X(a_i, t) := 1 \tag{1}\\ (X(a_i,t) = 0) \wedge (t \leq eet_i) &\Rightarrow (s(a_i) \geq t) \tag{2}\\ (X(a_i,t) = 0) \wedge (lst_i \leq t) &\Rightarrow (e(a_i) \leq t) \tag{3}\\ X(a_i,t) = 1 &\overset{def.}{\Leftrightarrow} s(a_i) \leq t \leq e(a_i) \notag \end{align}$$

The above definition of the timetable-constraint includes the unary case for $cap(R_j) = 1$ as well as the cumulative case for $1 \leq n = cap(R_j)$ and imposes bounds consistency on

$$\forall t \in [0, T_{max}] : \sum_{a_i} cap(a_i, t, R_j) \leq cap(R_j)$$

Further, $X(a_i, t)$ is an implicit boolean variable denoting that a_i is scheduled at time t in the global timetable. Assuming that a test for $<, =, >$ as well as updating the variable domains D_i is performed in $O(1)$, the propagation rules (1-3) are constant time operations. By definition of T_{max} (cf. *section 2.2*) the global *timetable* can be setup incrementally in $O(n)$ using rule (1). Hence, the timetable constraint can be implemented in $O(n)$, which is given as worst-case complexity in [Bar04], where it is considered "a very fast algorithm" in practice.

Definition 4 (disjunctive [BLPN01]).
disjunctive*(Γ) $\hat{=}$ ensures that there is no overlap in time between any 2 activities in Γ and that $cap(R_j)$ is not exceeded:*

$$\begin{align} \neg A \wedge eet_i > lst_j &\Rightarrow e(a_j) \leq s(a_i) \tag{4}\\ \neg A \wedge eet_j > lst_i &\Rightarrow e(a_i) \leq s(a_j) \tag{5}\\ A &= cap(a_i, R_j) + cap(a_j, R_j) \leq cap(R_j) \notag \end{align}$$

The above definition imposes bounds consistency on the formula

$$cap(a_i, R_j) + cap(a_j, R_j) \leq cap(R_j) \vee e(a_i) \leq s(a_j) \vee e(a_j) \leq s(a_i)$$

which shows that the disjunctive constraint easily extends to the cumulative case. Furthermore, the propagation rules (4-5) state the denotational semantics of this constraint formulating the local non-overlapping property that a_i precedes a_j or vice versa. Regarding the unary case, this local property can be stated using the disjunction

$$e(a_i) \leq s(a_j) \vee e(a_j) \leq s(a_i)$$

As $|\Gamma| = n$, a naive implementation of the disjunctive constraint would consider $\binom{n}{2}$ disjunctions resulting in posting $O(n^2)$ local constraints. Clearly, this is not the desired result, that we expect from global propagation. In fact, the literature

[LN02] provides a worst-case complexity of $O(n)$ for the disjunctive constraint based on the above rule set, but as there is no algorithm given, the use of the propagation rules (4-5) cannot be guaranteed. Nevertheless, the disjunctive constraint outlines the advantage of global constraint propagation: one single constraint considering all problem variables at once, i.e. the variables encoding the set Γ, yields a more efficient algorithm ($O(n)$) than mimicking a global constraint using local propagation for each of the local properties ($O(n^2)$).

Definition 5 (edgefinding [BLPN01]).
edgefinding*(Γ) $\hat{=}$ determines, whether an activity $a_i \in \Gamma$ can be scheduled first ($a_i \prec \Omega$) or last ($\Omega \prec a_i$) among $\Omega \subseteq \Gamma$:*

$$\forall \Omega : \forall a_i \notin \Omega : [d_\Omega - r_{\Omega \cup \{a_i\}} < p_\Omega + p_i] \Rightarrow \Omega \prec a_i \tag{6}$$

$$\forall \Omega : \forall a_i \notin \Omega : [s(a_i) \geq max_{\emptyset \neq \Omega' \subseteq \Omega}(r_{\Omega'} + p_{\Omega'})] \overset{def.}{\Leftrightarrow} \Omega \prec a_i$$

$$\forall \Omega : \forall a_i \notin \Omega : [d_{\Omega \cup \{a_i\}} - r_\Omega < p_\Omega + p_i] \Rightarrow a_i \prec \Omega \tag{7}$$

$$\forall \Omega : \forall a_i \notin \Omega : [e(a_i) \leq min_{\emptyset \neq \Omega' \subseteq \Omega}(d_{\Omega'} - p_{\Omega'})] \overset{def.}{\Leftrightarrow} a_i \prec \Omega$$

where $r_\Omega = min_{i \in \{1,\ldots,n\} \wedge a_i \in \Omega} r_i$ denotes the minimum release date of an $a_i \in \Omega$, $d_\Omega = max_{i \in \{1,\ldots,n\} \wedge a_i \in \Omega} d_i$ denotes the maximum deadline of an $a_i \in \Omega$, and $p_\Omega = \sum_{i=1 \wedge a_i \in \Omega}^{n} p_i$ is the sum over all processing times of all activities in Ω. Propagation is done by checking, whether the execution of a_i and Ω in the interval $[r_{\Omega \cup \{a_i\}}, d_\Omega]$ (in $[r_\Omega, d_{\Omega \cup \{a_i\}}]$ respectively) violates the deadline of some activity $a_v \in \Omega$. If so, rule (6) deduces that $\Omega \prec a_i$ ($a_i \prec \Omega$ by rule (7) respectively). The edgefinding constraint extends to the cumulative case by checking additionally that $cap(R_j)$ is not violated in the intervals defined as above (rules 8-9). Formally, this is done by introducing the notion of the energy of a_i, $en_i = cap(a_i, R_j) \cdot p_i$, and of the energy of Ω, $en_\Omega = \sum_{a_i} en_i$ instead of the respective processing times p_i and p_Ω:

$$\forall \Omega : \forall a_i \notin \Omega : en_{\Omega \cup \{a_i\}} > cap(R_j) \cdot (d_\Omega - r_{\Omega \cup \{a_i\}}) \Rightarrow \Omega \prec a_i \tag{8}$$

$$\begin{aligned} \forall \Omega : \forall a_i \notin \Omega : max_\alpha max_\beta \, (r_{\Omega'} + \lceil rest(\Omega', c_i)/c_i \rceil) \leq s(a_i) &\overset{def.}{\Leftrightarrow} \Omega \prec a_i \\ \alpha := \Omega \subseteq \{a_1, \ldots, a_n\} : a_i \notin \Omega \wedge cap(R_j) \cdot (d_\Omega - r_{\Omega \cup \{a_i\}}) &< en_{\Omega \cup \{a_i\}} \\ \beta := \Omega' \subseteq \Omega : \Omega' \neq \emptyset \wedge rest(\Omega', c_i) &> 0 \end{aligned}$$

$$\forall \Omega : \forall a_i \notin \Omega : en_{\Omega \cup \{a_i\}} > cap(R_j) \cdot (d_{\Omega \cup \{a_i\}} - r_\Omega) \Rightarrow a_i \prec \Omega \tag{9}$$

$$\begin{aligned} \forall \Omega : \forall a_i \notin \Omega : min_\alpha min_\beta \, (d_{\Omega'} - \lceil rest(\Omega', c_i)/c_i \rceil) \geq e(a_i) &\overset{def.}{\Leftrightarrow} \Omega \prec a_i \\ \alpha := \Omega \subseteq \{a_1, \ldots, a_n\} : a_i \notin \Omega \wedge cap(R_j) \cdot (d_{\Omega \cup \{a_i\}} - r_\Omega) &< en_{\Omega \cup \{a_i\}} \\ \beta := \Omega' \subseteq \Omega : \Omega' \neq \emptyset \wedge rest(\Omega', c_i) &> 0 \end{aligned}$$

where $c_i = cap(a_i, R_j)$ and $rest(\Omega, c_i) = en_\Omega - (cap(R_j) - c_i)(d_\Omega - r_\Omega)$ denotes the remaining energy in Ω that can be scheduled without making the values in $D(s(a_i))$ inconsistent. Whereas the implementations in [BLPN01] are running in $O(n^2)$ or even $O(n^3)$, the best propagation algorithm for the edgefinding constraint in the literature is given by *Carlier and Pinson* and runs in $O(n \cdot log(n))$ time and $O(n)$ space.

Definition 6 (notfirstnotlast [BLPN01]).
notfirstnotlast*(Γ) $\hat{=}$ deduces, whether an activity a_i cannot be scheduled first ($\neg a_i \prec \Omega$) or last ($\neg\Omega \prec a_i$) among a set of activities $\Omega \subseteq \Gamma$:*

$$\forall\Omega : \forall a_i \notin \Omega : [d_\Omega - r_i < p_\Omega + p_i] \Rightarrow \neg(a_i \prec \Omega) \tag{10}$$
$$\forall\Omega : \forall a_i \notin \Omega : \neg(a_i \prec \Omega) \Rightarrow s(a_i) \geq eetmin_\Omega$$
$$\forall\Omega : \forall a_i \notin \Omega : [d_i - r_\Omega < p_\Omega + p_i] \Rightarrow \neg(\Omega \prec a_i) \tag{11}$$
$$\forall\Omega : \forall a_i \notin \Omega : \neg(\Omega \prec a_i) \Rightarrow e(a_i) \leq lstmax_\Omega$$

where $eetmin_\Omega = min_{i\in\{1,\ldots,n\}\wedge a_i\in\Omega} eet_i$ denotes the smallest of the earliest end times in Ω and $lstmax_\Omega = max_{i\in\{1,\ldots,n\}\wedge a_i\in\Omega} lst_i$ denotes the greatest of the latest start times in Ω. Propagation of the abvove rule set checks, whether the execution of a_i and Ω in the interval $[r_i, d_\Omega]$ (in $[r_\Omega, d_i]$ respectively) violates the deadline of some activity $a_s \in \Omega$. Thereof it follows from rule (12), that $\neg(a_i \prec \Omega)$ ($\neg(\Omega \prec a_i)$ follows from rule (13) respectively). Like the edgefinding constraint, the extension of the notfirstnotlast constraint to the cumulative case is straightforward. We just check that neither the deadlines are violated nor the maximum resource capacity $cap(R_j)$ is exceeded, which is realised by using the energy notion en_Ω as for the cumulative edgefinding constraint and considering the resource usage of a_i over the interval $[r_\Omega, min(eet_i, d_\Omega)]$ in rule (12) ($[max(lst_i, r_\Omega), d_\Omega]$ in rule (13)respectively) and by taking into account the available energy of R_j over the interval $[r_\Omega, d_\Omega]$:

$$\begin{aligned}
\forall\Omega : \forall a_i \notin \Omega : \alpha \;&\Rightarrow\; \neg(a_i \prec \Omega) && (12)\\
\alpha := [r_\Omega \leq r_i < eetmin_\Omega] \;&\wedge\; [en_\Omega + cap(a_i, R_j)(min(eet_i, d_\Omega) - r_\Omega) &&\\
&\quad > cap(R_j)(d_\Omega - r_\Omega)] &&\\
\forall\Omega : \forall : a_i \notin \Omega : \neg(a_i \prec \Omega) \;&\overset{def.}{\Leftrightarrow}\; max_{\Omega\subseteq\{a_1,\ldots,a_n\}:\alpha} eetmin_\Omega \leq s(a_i) &&\\
\forall\Omega : \forall : a_i \notin \Omega : \beta \;&\Rightarrow\; \neg(\Omega \prec a_i) && (13)\\
\beta := [lstmax_\Omega < d_i \leq d_\Omega] \;&\wedge\; [en_\Omega + cap(a_i, R_j)(d_\Omega - max(lst_i, r_\Omega)) &&\\
&\quad > cap(R_j)(d_\Omega - r_\Omega)] &&\\
\forall\Omega : \forall : a_i \notin \Omega : \neg(\Omega \prec a_i) \;&\overset{def.}{\Leftrightarrow}\; min_{\Omega\subseteq\{a_1,\ldots,a_n\}:\beta} lstmax_\Omega \geq e(a_i) &&
\end{aligned}$$

As mentionned in [Bar04], a propagation algorithm for the notfirstnotlast constraint runs in $O(n^2)$.

3 Specialised Branching

Usually, when it comes to constraint distribution in CP, it is clear, that standard branching and search techniques yield the desired result of a naive distribution or a first-fail distribution, i.e. to find the earliest possible branch in the search tree, that is a failure state and as a such can be omitted. Typically is the use of a naive distribution, i.e. select always the leftmost undetermined variable next, or the use of first-fail strategies like minimal size, minimal difference between domain bounds, the most constrained variable or other strategies. As far as it comes to the CSP-formulation of a scheduling problem, these standard branching schemes are insufficient, if they apply at all. Just like scheduling problems demand specialised scheduling constraints (cf. *section 2.3*), they also demand specialised branching schemes [LN02]. As the CSP-model for scheduling problems focuses on activities, so do the branching methods, that are being applied. One branching possibility is *first/last*-branching. This scheme takes on activity a_i from an unordered set of activities Ω and branches on $[first(a_i)] \vee [\neg first(a_i)]$ or $[last(a_i)] \vee [\neg last(a_i)]$. A different possibility of branching is the so-called *activity pair ordering.* In this branching, two unordered activities a_i and a_j on the same resource R_j are selected and the branching is performed on: $[e(a_i) \leq s(a_j)] \vee [e(a_j) \leq s(a_i)]$ A further branching method is the *minimal slack*- or *resource slack*-heuristic. In this context, the notion of a *slack* is defined as follows:

$$slack(a_i \prec a_j) = max(end(a_j) - min(s(a_i)) - p(\{a_i, a_j\})$$

i.e. a slack denotes the amount of free resource between the earliest end time and the latest starting time of two succeeding activities. To choose a minimal slack is important, since a minimal slack indicates, that there are minimal combinatorial possibilities to order these two activities, hence this is a critical position in the whole schedule. This heuristic for a pair of two activities easily extends to sets:

$$slack(\Omega) = max(end(\Omega) - min(s(\Omega)) - p(\Omega)$$

where the focus lies on the resource with minimal slack.

4 Summary

In this paper we introduced and defined the domain of *scheduling* and gave a formal classification of *scheduling problems.* Furthermore we introduced *constraint-based scheduling* as a solution method for those scheduling problems incorporating techniques of *constraint programming.* In this context an encoding of non-preemptive scheduling problems into a *constraint satisfaction problem* was presented and we provided a formal insight into the propagation rules of specialised global scheduling constraints being a fundamental part of this CSP-model. In combination with specialised branching strategies we actually saw that scheduling constraints are powerful global specialised constraints and that there are effcient global propagation algorithms implementing the presented rule sets. Thus we can conclude that based on those scheduling constraints, CBS is a highly efficient and very useful tool to solve combinatorial hard scheduling problems.

5 References

[Apt03] Krzystof R. Apt. *Principles of Constraint Programming.* Cambridge Univ. Press, Cambridge, 2003.

[Bak74] K.R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New-York, 1974.

[Bar04] Roman Barták. Constraint Satisfaction for Planning and Scheduling. In *14th International Coneference on Automated Planning and Scheduling*, Whistler, Canada, 2004.

[BLPN01] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling.* Kluwer Academic Publishers, Dordrecht, 2001.

[Bru98] Peter Brucker. *Scheduling algorithms.* Springer, Berlin, 2nd. rev. and enlarged ed. edition, 1998.

[cla] [http://wwwmayr.informatik.tu-muenchen.de/scheduling/] Scheduling Classification.

[Fro01] Markus P.J. Fromherz. Constraint-Based Scheduling. In *American Control Conference*, Arlington, VA, USA, June 2001.

[Kri04] Dr. Axel Krings. *CS 448/548: Survivable Systems and Networks.* University of Idaho, lecture 24 (03/12/04): Scheduling issues, 2004. [Course homepage] http://www.cs.uidaho.edu/~krings/CS448/index.html.

[LdB04] Matthijs Leendert den Besten. *Simple Metaheuristics for Scheduling, PhD Thesis.* Darmstadt University, Darmstadt, Germany, 2004.

[Leg04] Arnaud Legrand. *Introduction to Scheduling Theory.* Laboratoire Informatique et Distribution, IMAG CNRS, France, lecture 11.09.2004, 2004. [invited talk at "CS260: Parallel Computation" SDSC at the University of San Diego, California] http://www.sdsc.edu/~allans/cs260/lectures/main.pdf.

[LN02] Philippe Laborie and Wim Nuijten. Constraint-Based Scheduling in an A.I. Planning & Scheduling Perspective. In *6th International Conference on AI Planning & Scheduling*, Tolouse, France, 2002.

[ORc] [http://www.mathematik.uni-osnabrueck.de/research/OR/class/] Examples for Scheduling Classification.

[Sha05a] N. Shaklevich. *OR 32: Scheduling: Models and Algorithms.* School of Computing, University of Leeds, lecture 5: Complexity, 2004/2005. [Course homepage] http://www.comp.leeds.ac.uk/or32/.

[Sha05b] N. Shaklevich. *OR 32: Scheduling: Models and Algorithms.* School of Computing, University of Leeds, lecture 1: Introduction, 2004/2005. [Course homepage] http://www.comp.leeds.ac.uk/or32/.

[vH05] Willem-Jan van Hoeve. *Operations Research Techniques in Constraint Programming, PhD Thesis.* Amsterdam University, Amsterdam, Netherlands, 2005.