Constraints (2012) 17:51–85
DOI 10.1007/s10601-011-9115-6

SURVEY

Optimal methods for resource allocation and scheduling: a cross-disciplinary survey

Michele Lombardi · Michela Milano

Published online: 11 January 2012

Abstract Classical scheduling formulations typically assume static resource requirements and focus on deciding when to start the problem activities, so as to optimize some performance metric. In many practical cases, however, the decision maker has the ability to choose the resource assignment as well as the starting times: this is a far-from-trivial task, with deep implications on the quality of the final schedule. Joint resource assignment and scheduling problems are incredibly challenging from a computational perspective. They have been subject of active research in Constraint Programming (CP) and in Operations Research (OR) for a few decades, with quite difference techniques. Both the approaches report individual successes, but they overall perform equally well or (from a different perspective) equally poorly. In particular, despite the well known effectiveness of global constraints for scheduling, comparable results for joint filtering of assignment and scheduling variables have not yet been achieved. Recently, hybrid approaches have been applied to this class of problems: most of them work by splitting the overall problem into an assignment and a scheduling subparts; those are solved in an iterative and interactive fashion with a mix of CP and OR techniques, often reporting impressive speed-ups compared to pure CP and OR methods. Motivated by the success of hybrid algorithms on resource assignment and scheduling, we provide a cross-disciplinary survey on such problems, including CP, OR and hybrid approaches. The main effort is to identify key modeling and solution techniques: they may then be applied in the construction of new hybrid algorithms, or they may provide ideas for novel filtering methods (possibly based on decomposition, or on alternative representations of the domain store). In detail, we take a constraint-based perspective and, according to the equation CP = model + propagation + search, we give an overview of state-of-art models, propagation/bounding techniques and search strategies.

M. Lombardi (✉) · M. Milano
DEIS, University of Bologna, Viale del Risorgimento 2, 40136 Bologna, Italy
e-mail: michele.lombardi2@unibo.it

M. Milano
e-mail: michela.milano@unibo.it

Keywords Scheduling · Resource allocation · Constraint programming · Operations research · Hybrid algorithms

1 Resource allocation and scheduling

Following a widely accepted definition, scheduling involves allocating scarce resources to activities over time. Classical scheduling formulations[1] typically assume static resource requirements and focus on deciding when to start the problem activities, so as to optimize some performance metric. Sometimes, however, the decision maker has the ability to choose the resources for the execution of each problem activity. This is a far-from-trivial task, with deep implications on the quality of the final schedule. Despite being considerably less popular than classical scheduling in the research literature, joint resource assignment and scheduling problems are very frequent in practice: according to [11, 51], many real world settings feature *optional activities* which can be left non-executed (usually with an impact on costs) or *alternative recipes* to execute an activity, with each recipe corresponding to a possible assignment of execution resources or to a different set of sub-activities.

Here, we take into account a variety of scheduling problems, having as a common feature the fact that the resource assignment and resource requirement for each activity are decision variables. Reported applications include batching and scheduling in chemical plants with several reactors [86]; scheduling aircraft landings to multiple runways [10]; simultaneous scheduling of maintenance activities requiring different skills [14]; compilation of computer programs on heterogeneous Very Large Instruction Word (VLIW) architectures [61]. Problems of this class do also arise in the optimization of parallel applications on multi-core platforms [53].

Scheduling problems are well-known to be NP-hard and computationally challenging: not surprisingly, introducing resource assignment decisions drastically increases the time for solving the problem. As an example, the well-known PSPLIB benchmarks [50] include classical scheduling problems (Resource Constrained Project Scheduling Problems—RCPSP) as well as problems involving resource assignment decisions (Multi-mode Resource Constrained Project Scheduling Problems—MRCPSP): while RCPSP instances with up to 120 activities are solved by powerful hybrid CP/SAT methods [75], finding optimal solutions to instances with up to 30 activities of the MRCPSP is still considered a very challenging task [94].

Constraint Programming (CP) can claim a success story with pure scheduling [3], providing an elegant and versatile modeling framework, based on effective and efficient filtering algorithms and search strategies. Significant effort has been put into extending CP models and solution techniques for allocation and scheduling problems. Unfortunately the obtained results are not comparable to those of pure scheduling. The main reasons are the very large search space and lack of effective techniques for joint filtering of assignment and scheduling variables.

Operations Research (OR) scientists have also considered resource allocation and scheduling problems (see the survey by Brucker at al. [18]), mainly under the flag of the MRCPSP, introduced in the late 70's [31]. Taking advantage of a strong

[1] E.g. the Resource Constrained Project Scheduling Problem or Job Shop Scheduling.

mathematical basis, they have identified relevant problem properties and powerful optimization-based filtering rules (dominance rules). Nevertheless, no sharp dominance exists between OR and CP techniques: both approaches report individual successes, but they overall perform equally well or (from a different perspective) equally poorly.[2]

Starting from 2001, hybrid CP/OR approaches have been applied to resource allocation and scheduling problems: to the best of the authors' knowledge, works [41, 48, 85] are the earliest, most relevant examples; the mentioned approaches are mainly based on Logic Based Benders Decomposition (introduced in [46] and formalized in [45]) and report orders-of-magnitude speed-ups compared to pure CP and pure OR methods. Intuitively, since allocation and scheduling problems result from the composition of an assignment and a scheduling component, hybrid algorithms have the opportunity to use the most effective (heterogeneous) technique to target each problem part (e.g. Mixed Integer Linear Programming—MILP—for the assignment and CP for scheduling).

Motivated by the success of hybrid methods on resource assignment and scheduling, we provide a cross-disciplinary survey on the topic, including CP, OR and hybrid approaches. *The main effort is to identify key modeling and solution techniques, so that they may be applied in the construction of new hybrid algorithms, or they may provide ideas for devising novel filtering methods* (e.g. based on decomposition or on alternative representations of the domain store). In detail, we take a constraint-based perspective and, according to the equation CP = model + propagation + search, we give an overview of state-of-art models, propagation/bounding algorithms and search strategies.

In particular, we focus on approaches making some use of tree search, as the techniques they apply are more easily portable in a CP context. This choice excludes local search, greedy heuristics and all meta-heuristic methods; due to the problem complexity, those are however very important approaches: [88] reports a comprehensive list of previous heuristic and meta-heuristic approaches for the MRCPSP and describes a novel genetic algorithm; [36] describes a metaheuristic based on smart neighborhood functions; [53] is a good starting point for heuristics to map parallel programs on multi-core platforms. Self-adaptive Large Neighborhood Search has been successfully applied to scheduling problems [56] and to joint allocation and scheduling problems modeled via time interval variables [55] (see Section 3.1.2). Finally, some combinatorial problems such as Capacitated Vehicle Routing with Time Windows are related to resource allocation and scheduling, but are left out of this survey: for more details, the interested reader may refer to [87].

The outline of the paper is as follows: Section 2 introduces the considered problem class and relevant variants; Section 3 discusses the main modeling techniques in OR and CP, including decomposition based models. Sections 4 and 5 are devoted to filtering algorithms and bounding rules; an overview of search strategies is given in Section 6, while solution methods for decomposition based approaches are discussed in Section 7.

[2] In particular, both approaches have serious scalability issues [48], typically worse for OR methods (see e.g. the state of the art results reported in [39, 94]). On the other side, pure CP approaches have serious difficulties with important classes of objective functions such as tardiness costs (as hinted by the lack of research works on the topic).

2 Reference problem class

In this section, we provide a formal definition for the main problem class we take into account; relevant variants are discussed in Section 2.1.

We consider an allocation and scheduling problem defined over a set A of activities (say $\{a_0, a_1, \ldots\}$), representing atomic operations. Activities have temporal dependencies, described via a set E of pairs (a_i, a_j), representing end-to-start precedence relations. Activities and dependencies together form a directed acyclic graph $G = \langle A, E \rangle$, referred to as *project graph* following the OR literature. Without loss of generality, we assume there is a single source activity and a single sink activity. We refer to the start time of activity a_i as s_i (i.e. the first time instant t where the activity is executing) and to the end time of a_i as e_i (i.e. the first time instant where the activity is not executing, after it has started), with $s_i, e_i \geq 0$; the notation $\overline{s}/\overline{e}$ refers to the full vector of start/end times.

The problem defines a set R of resources r_k, with fixed capacity c_k over time (*renewable resources*, introduced in [31]). Those may represent, e.g., manpower, industrial machines or computing devices. We provide a formal description of allocation decisions by introducing binary variables x_{ik}, with $x_{ik} = 1$ if a_i is using r_k. Problem dependent constraints typically restrict the possible resource assignment choices, i.e. the allowed tuples for the full vector of allocation variables $\overline{x}$. The exact amount of each requirement depends on the assignment decisions: formally, this is a function $rq_{ik}(\overline{x})$, with $rq_{ik}(\overline{x}) \geq 0$. Similarly, the activity durations are allocation dependent, i.e. each duration is a function $d_i(\overline{x})$, with $d_i(\overline{x}) \geq 0$. Note the approach allows duration and requirement values to depend on allocation choices involving multiple activities.

A solution of the problem is a full assignment of start times s_i and allocation variables x_{ik} such that neither temporal nor resource capacity restrictions are violated. A solution is optimal if it has (with no loss of generality) the minimal feasible value for a given performance metric $F(\overline{x}, \overline{s}, \overline{e})$, depending in general both on allocation and scheduling choices. Finding an optimal solution is an NP-hard problem (the proof follows from the one in [51]).

2.1 Relevant problem variants

Resource assignment and scheduling problems mainly arise in practical contexts. As a consequence, the basic structure outlined in the previous section is often complicated by a number of side constraints and unique features. Rather than attempting a comprehensive classification, we describe some of the main variants encountered in the literature and considered in this paper.

2.1.1 Resource related variants

Besides renewable resource, other resource types are found in the literature; here, we mention the main ones.

Non-renewable resources Non-renewable resources have a starting availability and are consumed throughout the whole scheduling horizon, until depleted: project budget is a good example. They were introduced in [31] and are typically considered by MRCPSP approaches. If non-renewable resources are taken into account, finding

a feasible resource assignment is an NP-complete problem [51] and the whole optimization process becomes considerably harder (see [79] for some comments). The so-called *doubly constrained resources* (coupling a fixed capacity over time and an overall usage limit) are considered in some works, but can be modeled by combining a renewable and a non-renewable resource.

Variable capacity and requirements over time Resources with time *varying capacity* are taken into account in [79, 81]. Work [8] shows a technique to turn variable capacities into constant ones, by introducing fictitious activities and fixing their position in the schedule by means of time windows or time lags (see Section 2.1.2); the method is described for the RCPSP, but can be easily generalized to take into account resource assignments. The important case of *resource breaks* (e.g. vacation time, machine maintenance downtime) and break-interruptible activities (i.e. that can be preempted by breaks) is considered in [19] in an OR context for the MRCPSP and in [1, 3] in CP. Time varying *resource requirements* are instead considered in [30] and in [70] in the context of alternative activities (see Section 3.1.2).

Time/resource trade-offs Some allocation and scheduling formulations take into account time/resource trade-offs, by allowing activities to assume a different duration, depending on the consumed amount of the selected resources. The MRCPSP is the most relevant example since activity modes may specify different requirement values rather than different mapping choices. In such a case, mapping decisions can no longer be formalized via the x_{ik} variables. In this paper, we consider problems with time/resource trade-offs, provided they also feature resource mapping choices: this leaves out pure trade-off problems with fixed resource mapping, such as the Discrete Time/Resource Trade-off Problem in [28].

2.1.2 Temporal constraint related variants

Temporal constraints other than simple end-to-start precedence relations are very common in the literature; here, we give a fairly complete overview of the reported cases.

Start/start, start/end, end/end precedence relations End-to-start relations can be generalized by introducing start-to-start, end-to-end and start-to-end precedence constraints. Those can be easily turned one into another, provided activities have fixed durations, say d_i (for example $e_j \geq e_i$ becomes $s_j + d_j \geq e_i$). The transformation method is described in [32] for the RCPSP, but applies to the problem from Section 2 provided $d_i(\overline{x})$ is constant, or whenever during search a full resource assignment becomes known. The multi-mode RCPSP with this kind of precedence relation is known as Generalized MRCPSP.

As a relevant remark, if precedence relations other than end-to-start are taken into account, assigning the shortest possible duration to an activity may result in an increase of the schedule makespan [32]. This may prevent the application of several dominance rules (see Section 5.2.2).

Generalized precedence relations and time windows This case subsumes the previous one; additionally, minimal and maximal time lags (say $\delta^{\min}$ and $\delta^{\max}$) label the

precedence relations and constrain the time distance between the involved activities: formally in case of an end-to-start arc, the produced schedule must satisfy $\delta^{\min} \leq e_j - s_i \leq \delta^{\max}$. Minimal time lags enforce a minimum separation restriction, maximal time lags may model "best before" constraints, occurring e.g. in chemical and food industries. The Multi-mode RCPSP with this type of precedence constraint is known as MRCPSP with Generalized Precedence Relations (GPR). Time windows, constraining the execution of activities a_i between a release time rs_i and a deadline dl_i are equivalent to minimal/maximal time lags from the source node.

For pure scheduling problems, a maximal time lag on an arc (a_i, a_j) can be converted into a *negative* minimal time lag on the complementary arc (a_j, a_i); the transformation follows trivially by the inequality $e_j - s_i \leq \delta^{\max}$ and is described in [8, 18]. As a consequence, a graph with maximal time lags typically contains cycles. A temporally feasible assignment of start times can be still obtained via shortest path algorithms (as shown for the Simple Temporal Problem [27]); finding a temporally and resource feasible schedule, however, is NP-hard even with very simple side constraints on the assignment variables. Moreover, taking into account Generalized Precedence Relations may prevent the application of many dominance rules, as described above.

In allocation and scheduling problems, time lags may depend on the resource assignment (e.g. mode dependent time lags in [39, 74]). In the context of optimization for parallel programs on multi-core systems, allocation dependent minimal time lags are typically used to model communication costs depending on processor assignment decisions [52]. Observe that, as long as some assignment decisions have to be taken, allocation dependent durations are equivalent to start-to-start precedence constraints with allocation dependent time lags; as a consequence, most resource assignment and scheduling problem do behave as containing Generalized Precedence Relations, at least for part of the search process.

Setup times Setup times are separation constraints between tasks mapped on the same resources, modeling, e.g., the time required to perform cleaning operations or to change tools before executing an activity a_i. Unlike allocation dependent minimal time lags, the involved activities need not to be connected by an arc in the project graph (i.e. their order is not a-priori fixed). If the setup time between activities a_i and a_j does not depend on the order they appear in the schedule, it may be incorporated in their execution time. Conversely, one speaks of *sequence dependent setup times*, a relevant case in practical applications: this is considered in [34] for scheduling with alternative resources, [10] for scheduling aircraft landings and [86] for a batching and scheduling problem from the chemical industry.

2.1.3 Objective function types

Time based objectives Time based objectives used in resource assignment and scheduling are an extension of those used for pure scheduling problems. A time based cost function only indirectly depends on allocation choices, i.e. $F(\overline{x}, \overline{s}, \overline{e}) = F(\overline{s}, \overline{e})$. Makespan (overall completion time) minimization is the most frequently occurring objective in the literature; in this case we have $F(\overline{x}, \overline{s}, \overline{e}) = \max_i e_i$. Equivalently, assuming a_{n-1} is the single sink activity, one can state $F(\overline{x}, \overline{s}, \overline{e}) = e_{n-1}$.

In a real-world context, where oversubscribed problems with tight time-windows often arise, tardiness based costs are also very popular; for an activity a_i with deadline dl_i, the tardiness is the activity delay w.r.t the deadline; this is defined as $TD_i(e_i) = \max(0, e_i - dl_i)$. A tardiness cost function has the form $F(\overline{x}, \overline{s}, \overline{e}) = \sum_i w_i \cdot TD_i(e_i)$, where weights w_i are introduced for each activity to account for non-uniform tardiness costs. This type of objective is considered in [43, 55] from a methodological perspective and in [10, 33, 86, 92] in an industrial context. Sometimes, there is a cost to be paid for early completions, with earliness being formally defined as $ER_i(e_i) = \max(0, dl_i - e_i)$. Earliness costs are considered in [86] (where they are due to inventory maintenance), in [10, 33] (in case of early aircraft landing) and in [55] (from a methodological perspective). A detailed list of possible problem objectives appears in [79]: most of them are time-based.

Resource based objectives We refer as resource based objectives to cost functions directly depending only on allocation choices, i.e. $F(\overline{x}, \overline{s}, \overline{e}) = F(\overline{x})$. The main example are costs to be paid for assigning single activities to resources or for choosing a specific activity mode: those include processing costs [48], energy dissipation over multi-core systems [73], or assignment costs in general [41, 85]. Costs associated to the assignment of activity *pairs* (similarly to those of a Quadratic Assignment Problem) appear instead in [16, 63] to model bus congestion in a multi-core system and are considerably more challenging from a computational standpoint.

Time and resource based objectives This class includes in the first place any combination (e.g via weighted sum) of time-based and resource-based objectives. Moreover, some cost functions depend inherently on allocation and scheduling decisions; this is the case for setup costs, since they are defined for activities assigned to the same resource: those are considered in [34], in [86] (where they are due to machine cleaning operations) and in [73] (where they are associated to processor frequency switching). A second unusual example are the rescheduling costs taken into account in [26] in the context of reactive scheduling.

Regular vs non-regular objectives The notion of *regular performance measure* is introduced in [72] for the RCPSP and extended in [81] to the multiple mode case. A cost function is regular if its value may only improve by reducing the end time e_i of some activity (without changing the activity mode—i.e. the resource assignment). This is an *extremely* important case, since with regular objectives the set of optimal solutions always includes a *tight* schedule (see the detailed discussion on the Left-shift Rule in Section 5.2.2): this property is used to devise powerful pruning rules. Non-regular objectives are however very common in practice and include earliness costs, setup costs and all the mentioned resource-based objectives.

3 Modeling techniques

Here, we present the main modeling techniques developed for the reference problem in CP (Section 3.1) and OR (Section 3.2); moreover, Section 3.3 discusses decomposition-based and hybrid approaches.

3.1 Constraint based models

3.1.1 Using classical activities and constraints

It is possible to use classical CP techniques for pure scheduling problems to take into account resource assignment decisions. In Constraint-based Scheduling [3], activities are represented by introducing integer variables for every activity a_i. In detail we have $\mathtt{S_i}$, representing the activity start time s_i and $\mathtt{E_i}$, representing the activity end time e_i. In assignment and scheduling problems, it is customary to introduce an integer variable $\mathtt{D_i}$ representing the activity duration; then the constraint $\mathtt{E_i} = \mathtt{S_i} + \mathtt{D_i}$ is enforced. Variables $\mathtt{S_i}$, $\mathtt{E_i}$ and $\mathtt{D_i}$ are assumed to have convex domains; their bounds are referred to by means of conventional names: $\min(\mathtt{S_i})$ is the Earliest Start Time—$EST(a_i)$—and $\max(\mathtt{S_i})$ is the Latest Start Time—$LST(a_i)$; the Earliest End Time and Latest End Time—$EET(a_i)$ and $LET(a_i)$—are defined analogously for the $\mathtt{E_i}$ variable.

End-to-start precedence relations are modeled as linear constraints $\mathtt{E_j} \leq \mathtt{S_j}$. Generalized Precedence Relations can be taken into account by posting constraints $\delta^{\min} \leq \mathtt{E_i} - \mathtt{S_i} \leq \delta^{\max}$, according to the definition from Section 2.1.2. Enforcing consistency for the resulting network is known as Simple Temporal Problem [27] and can be done via propagation of the inequality constraints. However, detecting an infeasibility with this method takes in general time proportional to the largest $\mathtt{S_i}/\mathtt{E_i}$ domain. Alternatively, precedence constraints can be explicitly stated (e.g. by means of global constraints) and one may use a shortest path algorithm for graphs with negative weights, such as Bellmann-Ford (with complexity $O(|A||E|)$, where $|A|$ is the number of activities and $|E|$ is the number of arcs).

Allocation decisions can be modeled [2] by introducing a binary variable $\mathtt{X_{ik}}$ for each x_{ik}; duration variables are linked to allocation decisions by a constraint $\mathtt{D_i} = d_i(\mathtt{X})$, where $\mathtt{X}$ represents the whole set of $\mathtt{X_i}$ variables and the function $d_i(\cdot)$ is the one from Section 2. Side constraints usually restrict the possible resource assignments. Resource restrictions are enforced via the *cumulative* constraint (see [3] for a reference). The effect of resource assignments can be modeled by using variable resource requirements: in detail, the amount by which activity a_i requires resource r_k is modeled via an auxiliary variable $\mathtt{RQ_{ik}}$; the variable value depends on allocation decisions, i.e. $\mathtt{RQ_{ik}} = rq_{ik}(\mathtt{X})$, where $rq_{ik}(\cdot)$ is the function from Section 2. Classical filtering for the cumulative constraint can be used in this case by assuming $\mathtt{RQ_{ik}}$ has minimal value.

$$
\begin{array}{rll}
\text{min:} & F(\overline{\mathtt{X}}, \overline{\mathtt{S}}, \overline{\mathtt{E}}) & \\
\text{subject to:} & \mathtt{E_i} = \mathtt{S_i} + \mathtt{D_i} & \forall a_i \in A \\
& \mathtt{E_i} \leq \mathtt{S_j} & \forall (a_i, a_j) \in E \\
& \mathtt{cumulative}(\mathtt{S}, \mathtt{D}, \mathtt{RQ_{ik}}, c_k) & \forall r_k \in R \\
& \mathtt{D_i} = d_i(\mathtt{X}) & \forall a_i \in A \\
& \mathtt{RQ_{ik}} = rq_{ik}(\mathtt{X}) & \\
\text{with:} & \mathtt{S_i}, \mathtt{E_i}, \mathtt{D_i} \in \{0, \ldots eoh\} & \\
& \mathtt{X_{ik}} \in \{0, 1\} &
\end{array}
$$

Fig. 1 A CP model for non-preemptive resource allocation and scheduling, with no side constraints

Figure 1 shows a basic CP model for resource allocation and scheduling with no side constraints; the cost function F is as from Section 2 and *eoh* denotes the largest considered time instant (End Of Horizon). Non-renewable resources are straightforward to model with this approach and result into a set of knapsack constraints on $\mathtt{X_{ik}}$ variables. Similarly, time-based objective and assignment costs are easy to deal with. Sequence dependent setup times are usually handled by introducing sequencing variables $\mathtt{B_{ij}}$ such that $\mathtt{B_{ij}} = 1$ iff a_i precedes a_j, i.e. we have $\mathtt{B_{ij}} = 1 \Leftrightarrow \mathtt{E_j} \geq \mathtt{S_i} + \delta_{ij}$, where δ_{ij} is the setup time. From a modeling perspective, there is hardly any allocation and scheduling problem beyond the reach of this approach. Unfortunately, resource and temporal propagation tend to be very weak when requirement or duration variables are unbound, so that more structured approaches are in practice needed.

3.1.2 Specific constraint based modeling techniques

Alternative resources This approach models assignment choices by allowing an activity a_i to require exactly one resource out of a set, say $R' \subseteq R$, of alternative resources [34, 66]; formally, relation $\sum_{r_k \in R'} x_{ik} = 1$ must hold. Typically, the whole resource set R is partitioned into subsets $R^0, R^1, \ldots$ of alternative resources. A fixed requirement (say ρ) is specified for the whole set, so that the corresponding $rq_{ik}(\mathtt{X})$ function is $\rho \cdot x_{ik}$. We say an alternative resource set R' is independent if a decision on R' has no impact on any other assignment choice, i.e. if there is no constraint connecting x_{ik} variables with $r_k \in R'$ to other variables in $\bar{x}$. From a modeling perspective, alternative resources represent a restriction compared to the approach in the previous section. However, they allow more powerful filtering, taking advantage of the fixed requirement value and the XOR relation between x_{ik} variables.

As a relevant special case, an independent set R' of n identical unary resources is equivalent to a single resource with capacity n (multi-capacity resource, see [65]): this representation results into huge search time savings whenever applicable. Unfortunately, side constraints or assignment dependent durations may make the resources non-identical and invalidate the equivalence. Setup times have the same effect, as they require to know the specific resource assignment for each activity.

Disjunctive temporal problem with finite domain constraints Assignment and scheduling problems over unary capacity resources can be modeled within the framework of the Disjunctive Temporal Problem with Finite Domain Constraints ($\mathrm{DTP_{FD}}$) by Moffitt, Peintner and Pollack [64]. A $\mathrm{DTP_{FD}}$ defines a network of time points, each with an associated temporal variable $\mathtt{T_i}$ and a finite domain variable $\mathtt{Y_i}$; time points can represents activity start/end events. It is possible to post between pairs of time points a *disjunction* of linear inequalities in the form $\mathtt{T_i} - \mathtt{T_j} \leq DB(\mathtt{Y_i}, \mathtt{Y_j})$; the value of the bounding function $DB(\mathtt{Y_i}, \mathtt{Y_j})$ depends on the finite domain variables. This approach exposes in depth the temporal structure of the problem, providing support for stronger filtering between time and assignment variables (see Section 4.1.1). Variables $\mathtt{Y_i}$ can be linked to assignment variables x_{ik} so that two activities are required not to overlap if they are processed by the same unary resource. The approach naturally handles setup times, by connecting sequencing variables (e.g. the $\mathtt{B_{ij}}$ we used before) to $\mathtt{Y_i}$ variables by means of

chaining constraints. Moreover, the DTP_{FD} can be used to model time/resource trade-offs (see Section 2.1.1).

Non-unary resources can in principle be handled by detecting possible resource conflicts at search time and consequently posting new disjunctions, similarly to what is done in many Precedence Constraint Posting approaches (see Section 3.3); this is however not reported in the literature.

Alternative activities The so-called Alternative Activities have been introduced by Beck and Fox in [11, 12]; they can be used to model resource assignment decisions and their impact on durations and requirements. The project graph is extended by including so-called *XOR nodes*; a pair of XOR nodes marks the start and the end of alternative "blocks", i.e. sets of alternative subgraphs. Formally, each activity is assigned an execution variable ex_i, such that $ex_i = 0$ if activity a_i does not execute, $ex_i = 1$ in case a_i executes and $ex_i \in \{0, 1\}$ as long as it is undecided. Successors/predecessors of a XOR node are mutually exclusive (i.e. $\sum_i ex_i = 1$). Alternative blocks can be nested, but this is the only allowed way to compose XOR nodes. A block with a single activity on each branch can be used to represent different execution recipes for the same logical activity (i.e. combinations of duration and resource requirements); the method is analogous to modes in the MRCPSP. The alternative recipes may require different resources or a different amount of the same resource; therefore this approach can be used to model time/resource trade-offs (unlike alternative resources). Alternative paths between nested XOR nodes may contain generic subgraphs rather than single activities; this allows one to model alternative process plans and makes the approach more expressive than the MRCPSP.

Alternative activities and plans are considered from a purely temporal perspective in Temporal Networks with Alternatives (TNA), introduced by Barták, Cepek and Surynek in [5]. TNAs feature so-called *branching nodes*, close in spirit to XOR nodes, but with no composition restriction. In principle, this allow the construction of non-nested alternative structures and model complex assignment constraints. Unfortunately, work [5] proves that completing a partial assignment of execution variables (e.g. finding a feasible resource assignment) is NP-complete in general, while the problem is polynomial in nested networks[3] [4].

In general, exposing assignment choices as alternative activities provides more information to filtering algorithms (see Section 4.1.1). As a main drawback, the project graph and the model become bigger. If Generalized Precedence Relations are not considered, modeling assignment decisions over independent sets of alternative resources requires an exponential number of activities (i.e. one for each combination of resource assignments). Conversely, by using GPRs it is possible to avoid the combinatorial blow-up; this can be done by (1) building an alternative block for each independent set of resources; (2) synchronizing their opening/closing XOR nodes using precedence constraints with time lags.

[3]We conjecture the result may hold for slightly more general graph structures, such as those considered in [63].

Optional activities and interval activities The so-called optional activities are similar to alternative activities since they have an associated execution variable ex_i; however, they are not necessarily part of an alternative block. They are introduced by Le Pape in [59], where the activity representation of ILOG-Scheduler is extended, so that a 0 duration denotes a non-executing activity. Optional activities requiring different resources can be used to model complex resource assignments decisions, by properly constraining their execution variables.

Building on ideas developed for alternative and optional activities, in [57] Laborie and Rogerie proposed to handle optional activities as first class variables, referred to as *time-interval variables*. A time interval variable $\underline{\mathtt{A}}_\mathtt{i}$ has values in the domain $\perp \cup \{[s, e) \mid s, e \in \mathbb{Z}, s \leq e\}$. In detail, either the variable is *non-executed* ($\underline{\mathtt{A}}_\mathtt{i} = \perp$) or its value is a half-open interval $[s, e)$ with integer bounds. Non-executed variables have no effect on the constraints they are involved in. Time interval variables can be connected by generalized precedence constraints, or can be organized in explicit alternative and hierarchical blocks; each block corresponds to a macro-interval $\underline{\mathtt{A}}_\mathtt{i}$, spanning over a set of (possibly alternative) sub-intervals $\underline{\mathtt{A}}_\mathtt{j}$.

So-called execution constraints $exec(\underline{\mathtt{A}}_\mathtt{i})$ force a variable to be executed and can be aggregated into unary or binary logical expressions (e.g $exec(\underline{\mathtt{A}}_\mathtt{i}) \Rightarrow exec(\underline{\mathtt{A}}_\mathtt{j})$). Resource constraints are taken into account in [58], where time-intervals are associated to step functions (referred to as cumul functions), representing the resource usage profiles. The approach is showcased in [55] on three practical examples.

Complex resource allocation and scheduling problems can be modeled by encoding resource assignment as execution decisions for interval variables.

The approach provides strong support for filtering algorithms thanks to the use of actual activities to expose different execution recipes, and thanks to explicitly stated alternative blocks. The availability of GPRs avoids the exponential growth of model size in case of independent sets of alternative resources. This approach is currently adopted by IBM-ILOG CP Optimizer and Google or-tools.

3.2 Mixed integer linear programming models

Most of the OR literature about resource allocation and scheduling is related to the Multi-mode Resource Constrained Scheduling Problem (MRCPSP), first introduced in [31]. In the MRCPSP, each activity a_i can be executed in one out of a *set of possible modes* M_i. Each mode $\mu_h \in M_i$ specifies a set of resource requirements and a duration value. The MRCPSP does not strictly fit the notation introduced in Section 2, but can be taken into account by introducing a mode variable m_i for each activity a_i and making the requirements and duration functions mode dependent. In practice, we can link the resource requirement and the activity duration to a single variable m_i representing the activity mode, rather than having a vector of 0–1 decision variables $\overline{x}$ representing the assignment/non-assignment of an activity to each resource.

The multi-mode formalism allows one to model time/resource trade-offs, since different modes may require the same resources in different amounts. Similarly to alternative activities and time interval variables, an exponential number of modes is required to model assignment decisions over independent sets of alternative

$$\min \quad F(\overline{\mathtt{E}}) \tag{1}$$

$$\text{s.t.} \sum_{\mu_h \in M_i} \sum_{t=0}^{eoh} \mathtt{E_{i,h,t}} = 1 \quad \forall a_i \in A \tag{2}$$

$$\sum_{\mu_h \in M_i} \sum_{t=0}^{eoh} t \cdot \mathtt{E_{i,h,t}} \leq \sum_{\mu_h \in M_j} \sum_{t=0}^{eoh} (t - d_j(\mu_h)) \cdot \mathtt{E_{j,h,t}} \quad \forall (a_i, a_j) \in E \tag{3}$$

$$\sum_{a_i \in A} \left[\sum_{\mu_h \in M_i} rq_{i,k}(\mu_h) \sum_{t=t_0}^{t_0+d_i(\mu_h)-1} \mathtt{E_{i,h,t}} \right] \leq c_k \quad \forall r_k \in R,\ \forall t_0 = 0 \ldots eoh \tag{4}$$

$$\mathtt{E_{i,h,t}} \in \{0, 1\} \quad \forall a_i \in A,\ \forall \mu_h \in M_i,\ \forall t = 0 \ldots eoh \tag{5}$$

Fig. 2 A time indexed model for non-preemptive MRCPSP

resources, unless GPRs are supported. The main Mixed Integer Linear Programming (MILP) models for the MRCPSP can be classified into *time indexed* and *disjunctive* and are described in the following.

3.2.1 Time indexed model

In time indexed models, binary variables $\mathtt{E_{i,h,t}}$ are introduced to denote whether an activity a_i is scheduled to *finish* at time t in mode μ_h. In Fig. 2, we report the model introduced by Talbot in [84]. Constraints (2) require each activity to be finished by the end of horizon. Constraints (3) enforce end-to-start precedence relations and constraints (4) resource capacity restrictions. The cost function has been re-formulated to take into account the different decisions variables.

Time indexed models allow resource capacity restrictions to be formulated as linear constraints. As a drawback, due to the use of a discrete time representation, the number of variables depends on the length of the horizon, resulting in limited scalability. In an attempt to address the issue, [93] proposes two alternative formulations, but reports no substantial improvement. With time indexed models it is straightforward to represent non-convex temporal domains; this allows to easily handle disjoint time windows, required e.g. for resource breaks with non-interruptible activities. Variable resource capacities over time are similarly easy to deal with (see [94]). The linear relaxation provided by a time indexed model is usually stronger than that of a disjunctive one, in particular in case of tight resource constraints.

Disjunctive model In a disjunctive model, a start variable $\mathtt{S_i}$ is introduced for each activity a_i. Mode assignments are represented by variables $\mathtt{M_{ih}}$, such that $\mathtt{M_{ih}} = 1$ iff activity a_i is executed in mode μ_h. A complete model (a slight elaboration over the one by Heilmann in [39]) is shown in Fig. 3. Constraints (7) represent end-to-start precedence relations and Constraints (9) force a mode to be assigned to each activity. The notation $A(\overline{\mathtt{S}}, t)$ refers to the set of tasks executing at time t, i.e. such that $\mathtt{S_i} \leq t < \mathtt{S_i} + \sum_{\mu_h \in M_i} d_i(\mu_h) \cdot \mathtt{M_{ih}}$. The cost function has been re-formulated to take into account the different decisions variables.

Disjunctive models require a smaller number of decision variables compared to time indexed ones. A linear representation for Constraints (8) can be provided

$$\min \quad F(\overline{\mathtt{S}}, \overline{\mathtt{M}}) \tag{6}$$

$$\text{s.t.} \quad \mathtt{S}_j - \mathtt{S}_i \geq \sum_{\mu_h \in M_i} d_i(\mu_h) \cdot \mathtt{M_{ih}} \qquad \forall (a_i, a_j) \in E \tag{7}$$

$$\sum_{a_i \in A(\overline{\mathtt{S}}, t)} \sum_{\mu_h \in M_i} rq_{i,k}(\mu_h) \cdot \mathtt{M_{ih}} \leq c_k \qquad \forall r_k \in R, \forall t = 0 \ldots eoh \tag{8}$$

$$\sum_{\mu_h \in M_i} \mathtt{M_{ih}} = 1 \quad \forall a_i \in A \tag{9}$$

$$\mathtt{S_i} \geq 0 \quad \forall a_i \in A \tag{10}$$

$$\mathtt{M_{ih}} \in \{0, 1\} \quad \forall a_i \in A, \forall \mu_h \in M_i \tag{11}$$

Fig. 3 A disjunctive model for non-preemptive MRCPSP

by preventing the overlapping execution of conflicting activities and requires: (1) to identify the sets of activities possibly causing an overusage (e.g. the Minimal Forbidden Sets described in Sections 6.1 and 6.2); (2) to add constraints forcing the disjoint execution of some of the activities in the sets. There are however two strong disadvantages: in first place, the number of Minimal Forbidden Sets (and therefore of additional constraints) is in general exponential in the size of the graph.[4] In second place, the disjunction constraints make use of a big-M linearization and lead to a poor relaxation based bound (considerably worse than time-indexed models). As a matter of fact, all approaches based on disjunctive models such as [39] rely on specific search strategies to take care of resource constraints (see Section 6).

As an alternative, [74] proposes a more compact model where resource allocation and scheduling is formulated as a constrained rectangular packing problem. Unfortunately, since rectangular packing and renewable resource restrictions are not fully equivalent [13], the method may miss feasible/optimal solutions.

3.3 Hybrid and decomposition based approaches

Resource assignment and scheduling problems result from the composition of a pure assignment and a pure scheduling component: this hybrid nature can be exploited to split the overall problem into distinct stages with separated (although dependent) variables. Typically, one is left with a resource assignment problem (so-called master problem) and a pure scheduling problem with the allocation provided by the master (so-called subproblem).

Such a decomposition has a number of advantages: (1) models for both the stages are typically much smaller and have a cleaner structure, making their solution more efficient. For instance, it is possible to remove big-M expressions from a MILP model, improving the linear relaxation [48]. (2) Heterogeneous techniques can be used to

[4] It is quadratic if only unary resources are taken into account, making this approach fairly popular in such context.

solve the stage they are best for (e.g. MILP for the assignment and CP for scheduling) obtaining so-called hybrid approaches. (3) The resulting "pure" problems can be tackled with a much better established pool of techniques, such as those in [3, 20]. (4) In case of resource-based objectives (see Section 2.1.3), the cost function depends only on the assignment stage, with the scheduling one acting as a feasibility check, e.g. [16, 41, 48, 63]. (5) Finally, a clever decomposition may allow the scheduling stage to be split into a collection of independent subproblems, making its solution even simpler. This is relatively easy if the Project Graph contains no precedence relations or in case only setup times are specified, as shown in [21, 41]). Splitting the subproblem may be however be impossible if no special precedence constraint structure is assumed.

Alternatively, a problem decomposition can be obtained by separating constraints rather than problem variables. In such a case the master is an optimization problem, possibly formulated with some technique with no support for resource constraints; the subproblem is either a consistency check or a very simple feasibility problem. The approach is discussed from a general perspective in [40], and in [85] in the context of the Branch-and-Check framework. Typically, resource capacity constraints can be considered only in the subproblem, leaving temporal and assignment constraints in the master. This is a key idea in Precedence Constraint Posting approaches: in this case possible resource conflicts in the current schedule are identified at search time (see Sections 6.1 and 6.2) and resolved by adding cuts. These concepts have been first introduced in [47], further elaborated in [77], and applied to allocation and scheduling problems in [12, 39]. The approach has been applied to scheduling problems by [71] and [54].

The obvious drawback of decomposition is the loss of valuable information due to the decoupling. In order to improve the solution quality, the two stages are solved iteratively in an interacting fashion: in particular, (1) cuts are injected in the assignment problem whenever the scheduling problem is solved (either successfully or not). Moreover (2) in all the mentioned references the assignment problem contains some relaxation of the scheduling problem constraints (so-called subproblem relaxation). Properly engineering the interaction between master and sub-problem is essential for the effectiveness of a decomposition based approach: systematic, optimal, methods to achieve this goal are formalized in the Branch-and-Check [85] framework and (most relevantly) in Logic-based Benders Decomposition (LBD), formally defined by Hooker and Ottosson in [45]. Both the approaches are discussed in Section 7.

The paper [86] tackles an allocation, batching and scheduling problem in the chemical industry introducing *temporal* decomposition: the time horizon is split into time buckets; an LBD approach is used to solve an allocation and scheduling problem for each bucket, with resource assignment acting as the master problem; the schedule for time bucket i is built by extending that of bucket $i-1$; solved buckets are never revised. Temporal decomposition is applied in a more systematic fashion by Coban and Hooker in [24]; however, this is done in the context of a single facility scheduling problem.

Finally, it is possible to devise hybrid approaches not relying on decomposition: work [48] reports a double CP/MILP model for an assignment and scheduling problem; specifically, branching is essentially based on the CP representation, while the MILP model is used to obtain bounds via its linear relaxation.

4 Propagation

In the context of resource assignment and scheduling problems, the key difficulty addressed by propagation techniques is to provide the tightest possible interaction between allocation and scheduling choices. In the following, we overview existing propagation techniques, roughly categorized in *temporal*, *logical* and *resource* filtering, depending on the aspect they mainly focus on. It is interesting to see how most of the filtering techniques for assignment and scheduling problems are in fact relatively simple extensions of the corresponding methods for pure scheduling: we found this stimulating rather than disappointing, as it leaves room for possible future improvements. With decomposition methods, classical scheduling constraints can be used in each subproblem; the problem of the interaction between resource assignment and scheduling translates instead to the problem of the interaction between the resulting components (addressed in Section 7).

4.1 Temporal reasoning

If only end-to-start precedence relations are defined, temporal constraint propagation can be done via the classical Critical Path Method [49]: lower bounds on the time windows (i.e. ESTs) can be obtained by assuming the durations are fixed to the lowest value and computing shortest paths from the source node. Temporal reasoning and time window determination have an important role in reducing the size of time-indexed models (see Section 3.2.1) and the quality of their linear relaxation.

If maximal time lags or precedence constraints other than end-to-start are considered, minimum durations do not necessarily lead to valid ESTs (this happens if a backward arc is part of the critical path as discussed in [32]). Correct time windows can be computed by relying on the equivalence between durations and min/max time lags (see Section 2.1.2), i.e. $min(d_i(\overline{x})) \leq e_i - s_i \leq max(d_i(\overline{x}))$: enforcing bound consistency on the resulting inequality constraints or performing shortest/longest path computation on the equivalent Simple Temporal Problem leads to valid EST/LET values. Obviously, no issue exists if durations are not allocation dependent; in general, however, when complex precedence constraints are taken into account, one should be aware that shortest durations do not necessarily result in the shortest possible schedule. As a remark, this prevents the application of some dominance rules (see Section 5.2.2).

4.1.1 Temporal reasoning with alternative resources and activities

Alternative and optional activities can be tackled by including the execution variables $ex_i \in \{0, 1\}$ in the precedence constraint expression, so that an end-to-start constraint becomes e.g. $ex_i \times ex_j \times s_j \geq ex_i \times ex_j \times e_i + d_i(\overline{x})$, as in [6]. The resulting propagation is however usually weak: this is quite expected since no specific constraint structure is assumed for the ex_i variables.

Stronger propagation can be obtained for nested alternative blocks [6, 12], for alternative groups of time-interval variables [57] and for alternative resources: the basic technique used to perform filtering is the same in all cases and is described

here in the context of alternative resources. In detail, let a_i be an activity requiring ρ units of one resource out of a set R' of m alternative resources, propagation can be performed as if a_i was split into m alternative activities $a_i^0, \ldots a_i^{m-1}$, each requiring ρ units of a specific resource in $r_k \in R'$. Then temporal reasoning maintains the constructive disjunction between the time window of the alternative sub-activities (see [34, 65]):

$$EST(a_i) = \min_{r_k \in R'}\{EST(a_i^k)\} \qquad LST(a_i) = \max_{r_k \in R'}\{LST(a_i^k)\}$$

$$EET(a_i) = \min_{r_k \in R'}\{EET(a_i^k)\} \qquad LET(a_i) = \max_{r_k \in R'}\{LET(a_i^k)\}$$

classical filtering methods for temporal constraints can be used to prune the time windows of the sub-activities. Whenever the domain of a start time variable becomes empty, the corresponding resource r_k is removed from the set of possible assignment choices, or the corresponding ex_i^k variable is set to 0 in case of alternative activities or time interval variables.

A similar approach is adopted by the least commitment method for the DTP_{FD} described in [64]. Here, temporal constraints with disjunctive bounds in the DTP_{FD} are associated with min/max time lags by convexification (i.e. by assuming the loosest bounds $DB(\mathtt{Y_i}, \mathtt{Y_j})$ allowed by the finite domain variables), then relaxed, convex, time windows are computed for each node. Whenever the time windows are tightened, incoherent $\mathtt{Y_i}$ values are ruled out.

As a last, very interesting case, the time-interval variable approach described in [57], allows *binary* logical constraints to be specified on the ex_i variables. Those are encoded as an implication graph, allowing joint logical and temporal propagation to be performed. In detail, assuming an arc (a_i, a_j) exists, whenever the logical network can infer the relation $ex_i \Rightarrow ex_j$ the arc can propagate the conditional bounds from s_j to e_i. Similarly, if the relation $ex_j \Rightarrow ex_i$ can be inferred, then the other half of the propagation can be performed. This allows propagation of temporal bounds even if the execution status is not yet fixed.

4.2 Logical filtering

Logical filtering includes the approach described in [7] for Temporal Network with Alternatives, that improves propagation by adding redundant side constraints on the execution variables. Such redundant constraints are based on the identification of classes of "equivalent" activities, inferred from the network structure. Equivalent activities share the same execution variable: explicitly stating this information substantially improves propagation over resource assignment variables. The paper proves that detecting all equivalent nodes in a network is NP-hard, but proposes three polynomial complexity methods to identify some equivalence relations. The most effective technique relies on the detection of nested substructures, in which case the entry and exit node can be considered equivalent. A second relevant example is the implication graph in [57], that may allow stronger propagation compared to local consistency.

4.3 Resource reasoning

The earliest edge-finding algorithm for resource assignment and scheduling problems is discussed in [2]: optional activities are tackled by assuming non-executed activities to have a duration of zero. Alternative resource assignments are dealt with by conditioning the resource requirement with the demand variables $\mathtt{x_{ik}}$ from Section 3.1. The resulting propagation is however quite weak.

Optional activities modeled via execution variables are first taken into account into edge-finding reasoning in [12]. In details, the algorithm is based on two simple ideas: (1) an optional activity a_i should not be taken into account when computing bounds on any other activity a_j; (2) if the time window of a_i gets empty due to resource propagation, the value 1 should be removed from the execution variable ex_i (i.e. the activity is deemed non-executable). This kind of reasoning to extend classical propagators for resource constraints to optional activities is the same encountered for temporal propagation in Section 4.1.1; it is also the key idea behind many other filtering algorithms, applied for example in [91] in the context of efficient edge finding with Θ-trees; in [34] for alternative resources, where the execution variables are replaced by resource assignment choices; in [90] for mixed timetable/edge-finding; in [89] within an interesting method to filter the maximal resource usage and maximal duration; in [70] where activities with negative resource consumption (i.e. resource producers) are taken into account.

The opposite process (deducing the necessary execution of an activity) is inherently more difficult. Typically, the most effective way to reach such a conclusion is propagating side constraints on the resource allocation (or the execution) variables. As an exception, in [70] it is shown how to deduce the necessary execution of activities with *negative* resource consumption.

For a set R' of alternative resources, complementary propagation is possible by introducing a redundant resource [65] with capacity $\sum_{r_k \in R'} c_k$ and applying classical filtering algorithms. Combined with the previously discussed methods, the approach provides improved propagation as long as the resource assignment is undecided.

5 Lower bounds and bounding rules

Research efforts in the context of the Multi-mode RCPSP have focused on bounding and dominance rules. Bounding rules refer to inferring necessary restrictions on time windows, mode assignments and scheduling decisions; they are closely related to filtering in CP. Dominance rules are based on some proof that the set of optimal solutions must include a schedule with specific properties, which are then used to narrow the search space. As an exception, some dominance rules exploit knowledge on past search to deduce that no better schedule can be reached from the current search node: they can be considered as a form of no-good learning. The most relevant bounding and dominance rules are presented in Section 5.2. Conversely, lower bounds for the MRCPSP tend to be not so effective: they are discussed in Section 5.1.

5.1 Lower bounds

Lower bounds on the optimal problem cost are a fundamental component of many search strategies (such as branch & bound, branch & cut, branch & price). In CP, a lower bound can be encapsulated in a global constraint, improving the effectiveness on optimization problems and providing access to useful information, such as reduced costs [35].

Effective lower bounds for resource allocation and scheduling problems are difficult to obtain. Time indexed and disjunctive models for the MRCPSP provide ready to use bounds via their linear relaxation, but those are unfortunately not so tight. The relaxation of time-indexed models provides the best results, due to lack of linearized constraints via big-Ms. To the authors knowledge, [94] is the only work directly exploiting a bound obtained by the relaxation of a (time-indexed) ILP model. the bounding technique is employed in two stages: in a preprocessing step to tighten time windows and reduce the model size, then during the solution process to fathom search nodes. The tightness of the linear relaxation can be improved by adding redundant cuts: this is done in [94] for the MRCPSP and in [10] in the context of scheduling aircraft landings.

In case of regular performance measures (see Section 2.1.3) a straightforward lower bound on the optimal cost can be obtained by ignoring resource constraints and starting each non-scheduled activity at the Earliest Start Time, obtained via longest path computation. Interestingly, this is the most widely employed bound for resource assignment and scheduling problems, since it is both easy to obtain and reasonably effective. In case of non-regular objectives, things get more complicated since an earliest start schedule does not necessarily correspond to a better cost: for earliness and tardiness objectives, [33] develops a modified network simplex algorithm which can be used to obtain in polynomial time an optimal schedule (without resource constraints). In [78] two families of lower bounds are proposed for the parallel machine scheduling problem in which the jobs have distinct due dates with earliness and tardiness costs: assignment-based lower bounds and bounds based on a time-indexed formulation of the problem. Lower bounds for sequence dependent setup times are considered in [34], while [26] discusses lower bounds for rescheduling costs.

An interesting technique to strengthen a lower bound consists in performing truncated tree search with a maximum depth: the weakest bound on the tree frontier is valid for the root node. The method is applied in [94] and [26].

5.2 Bounding rules

Bounding rules are tree reduction techniques that check if the current search node can be fathomed. Unlike filtering algorithms, bounding rules are executed as part of the search method and are not attached to a constraint. As a consequence, the rule formulation is tailored on a specific branching scheme (see Section 6.2), even though the main underlying idea usually has broader applicability. There is no general coordination mechanism (such as propagation), so that the combination of different rules is up to the algorithm designer. From a CP perspective, bounding rules may serve as a basis for the development of filtering algorithms.

5.2.1 Feasibility and symmetry based rules

Non-renewable resource rule This rule appears in [38], in [80] (with the name "non-renewable resource consumption") and in [26] (as "resource infeasibility rule"). The rule considers each *non-renewable* resource r_k: if scheduling each currently unscheduled activity in the mode with the lowest request for r_k would exceed its capacity c_k, then the current partial schedule cannot be feasibly completed. The rule is given a very efficient static formulation in [79] (where it is referred to as "input data adjustment"). This kind of reasoning is subsumed in CP by the usual constraint propagation techniques on the resource capacity constraints and the assignment variables.

Immediate selection We group here two different rules, sometimes also referred to as "Non-delayability", based on the principle that an obvious or forced scheduling choice should be immediately performed. The first rule, described in [18, 38, 79, 80] requires to identify an unscheduled activity a_j having fixed mode and no chance to overlap (due to resource or precedence constraints) with scheduled or unscheduled activities. In this case, a_i can be immediately selected for scheduling. The second rule only appears in [39] and specifies that, if a possible resource conflict (e.g. a Minimal Critical Set) admits a single resolution choice, than this should be immediately performed. The mentioned works contain specific adaptations to different scheduling schemes.

Single enumeration rule The single enumeration rule, introduced in [81] and further applied and refined in [38, 79] is a type of dynamic symmetry breaking constraint for precedence tree branching (see Section 6.2). The rule targets two activities a', a'', scheduled in two subsequent search steps i and $i+1$ in mode m' and m''; if their assigned start times do not depend on which activity is scheduled at step i and which one at $i+1$, then only one sequence needs to be considered.

5.2.2 Dominance rules

Dominance rules are a super set of symmetry breaking constraints for optimization problems. They are based on the observation that the set of optimal solutions necessarily contains a schedule with specific properties. One can therefore focus on the generation of that specific schedule, considerably narrowing the search space.

Dominance rules come with specific applicability assumptions. In particular, they are often restricted to regular cost functions (see Section 2.1.3). All presented dominance rules target a search process where a partial schedule is built by assigning a start time to unscheduled activities, proceeding in chronological order.

Left-shift rule This extremely important class of dominance rules is based on a property of regular objective functions (see Section 2.1.3), and on the concept of left-shift (discussed in details in [82]). A left-shift is an operation on a single activity a_i within a feasible schedule S, deriving a feasible schedule S', such that $e_i(S') < e_i(S)$ (where $e_i(S)$ is the end time of a_i in S) and no other schedule modification occurs. A left shift of exactly one time unit is called *local left shift*. A multi-mode left shift [80] is a left shift of a_i where the activity is allowed to change mode (e.g. to be assigned to a different resource). A schedule is *active* if it is feasible and no activity can be

left-shifted, a schedule is *tight* if it is feasible and no multi-mode left-shift can be performed. In case of regular performance measures (see Section 2.1.3), the set of optimal schedules is guaranteed to contain a tight schedule (an active schedule in the case of the RCPSP).

A pruning rule can be devised based on those properties; the general version of this left-shift rule states that a partial schedule in which an activity a_i can be left-shifted without violating the precedence and the resource constraints needs not to be completed (as it is dominated by another active or tight schedule). Several variants of the rule exist: they are employed in [26, 38, 79, 80]. Brief and effective descriptions are reported in [18].

Multi-mode rule This rule (employed in [18, 25, 38, 79, 80]) is based on the so-called mode-reduction operation. A mode reduction [80] of an activity a_i within a feasible schedule is an operation changing the mode of a_i to one with shorter duration, without changing its finish time and without violating the constraints or changing the modes or finish times of the other activities. A schedule is called *mode-minimal* if no mode reduction can be performed. If there is an optimal schedule for a given instance, then there is an optimal schedule which is *both tight and mode-minimal*. Some care must be observed with maximal time lags and precedence constraints other than end-to-start (see Sections 2.1.2 and 4.1). Note that a tight schedule may not be mode-minimal, and a mode-minimal schedule may not be tight (for an example see [80]). The rule states that, if a mode reduction can be performed on an activity a_i with e_i equal to the current scheduling time, then the current partial schedule needs not to be completed.

Order swap rule An order swap [18, 38] is an operation on a feasible schedule targeting two activities a_i, a_j with $j > i$, such that a_i, a_j are scheduled in sequence, i.e. $e_i = s_j$. The order swap consists of an exchange of the start time of the two activities, with no violation of a precedence or resource constraint. Changing the mode of any activity or the start time of any activity other than a_i and a_j is not allowed.

A schedule where no order swap can be performed is called *order monotonous*. If the order swap does not affect the objective function value (this is the case e.g. for the makespan, but not for tardiness costs), the set of order monotonous schedules is guaranteed to contain an optimal schedule. Therefore, before an activity a_i is scheduled at time t, if an order swap is allowed with any scheduled activity having end time t, then the current search node needs not to be further extended. The order swap rule is not the same as the single enumeration one, since the latter does not require the activities to form a sequence in the schedule.

5.2.3 Static bounding/dominance rules

Static bounding rules are introduced in [80] and used in most of the exact approaches for the MRCPSP developed later on. They are applied prior to the beginning of search and consist in the removal of *non-executable modes*, *inefficient modes* and *redundant resources*. The rule application is iterative, until a fix-point is reached, making them very similar to constraint propagation.

In detail, a mode μ_h for activity a_i is defined as non-executable if any of the corresponding resource requirements $rq_{ik}(\mu_h)$ exceeds the capacity of a renewable resource (or the capacity of a non-renewable resource, reduced by the sum of the

minimum requirements of all other activities). A non-renewable resource r_k is said to be redundant if its capacity exceeds the sum of the maximum consumption of all activities. Finally, a mode μ_h for activity a_i is inefficient if there exist some other mode μ_r with shorter duration and lower consumption for all resources. Note this is in fact a form of static dominance rule (see Section 5.2.2). As such, non-regular objectives, maximal time lags and precedence constraints other than end-to-start may prevent the rule application (see Section 2.1.3 and 4.1).

5.2.4 Multi-mode cutset rules

This family of rules requires to store information about past search. During the solution process, the current partial schedule is compared with the stored data. If the current partial schedule cannot be completed to a solution better than those obtained from a previously evaluated partial schedule, then backtracking is performed. The presented formulation is devised for makespan minimization, but does extend to regular performance measures [79]. Some care should be observed with Generalized Precedence Relations (see Sections 2.1.2 and 4.1). Cutset rules are described in [18, 25, 26, 38].

Given a partial schedule S, the cutset $C(S)$ is the set of activities scheduled so far. Besides the cutset, the rule requires to store the completion time of the partial schedule $e(S)$ (i.e. the highest end time among activities in S) and the leftover capacities of all non-renewable resources. Then:

Rule 1, dominated heads Let S be the partial schedule to be extended at time t in the current search step, having cutset $C(S)$. If a stored partial schedule S' exists, with the following features: (1) the same cutset, i.e. $C(S') = C(S)$; (2) lower or equal completion time, i.e. $e(S') \leq e(S)$; (3) higher or equal leftover capacities for all non-renewable resources; then the current partial schedule needs not to be completed.

A second rule is presented in [79] and provides a bound for the time span necessary to complete the current partial schedule. The rule requires to store, for each visited partial schedule S the updated latest finish time $LET(S, a_i)$ of a_i, after all possible continuations of S have been explored.

Rule 2, incompletable tails Let S be the partial schedule to be extended with activity a_i at time t in the current search step, having cutset $C(S)$. If a stored partial schedule S' exists, with the following features: (1) the same cutset, i.e. $C(S') = C(S)$; (2) higher or equal leftover capacities for all non-renewable resources; (3) $t + LET(S', a_i) - e(S') + 1 > LET(a_i)$; then the current partial schedule cannot be completed with a makespan better than the current $LET(a_i)$.

5.2.5 Effectiveness of the bounding rules

Some experimental evaluation of the described rules is reported in [26, 38, 79, 80]; additionally, [79] provides some details about the rule implementation. An overall thorough comparison is difficult, since different works have considered different bounding rules and sometimes targeted different instance sets. As a general remark, combining bounding rules is in general fruitful, i.e. there is not a sharp dominance relation.

The non-renewable resource rule is considered to be among the most effective techniques and provides the highest speed-up both in [80] and [79]; the reported improvement is less substantial, but still remarkable, in [26]. This result points out the difficulties encountered by OR methods when feasible schedules are not trivial to build (e.g. when non-renewable resource capacities are taken into account). The effectiveness of the left-shift rule is also well documented. Interestingly, the best results are usually obtained by testing the feasibility of local left-shifts. The single enumeration rule has a fundamental role within precedence tree branching [38]. The multi-mode rule performs nicely in the comparison from [80]. Among the cutset rules, the rule of dominated heads performs very well, definitely much better than incompletable tails. The immediate selection rule tends to be effective for small instances, but becomes more expensive and less likely to be triggered on larger ones. Static bounding rules are very effective for MRCPSP instances with high average resource requirements (specifically, Resource Strength—see [50, 80]). The order-swap rule introduced in [38] is as effective as the local left-shift one.

6 Search

The key problem addressed by search methods for assignment and scheduling problems is how to effectively explore a search space that is in the typical case impressively large: this is a combined effect due to the domain size of temporal variables and to the presence of resource allocation choices. Efficacy can be obtained by applying one of the following actions: (1) by guiding search as quickly as possible towards feasible/optimal solutions; (2) by designing branching decisions so as to maximize propagation effectiveness; (3) by reducing the portion of the search space that needs to be explored (e.g. via the application of dominance rules or dynamic symmetry breaking techniques). While powerful techniques to achieve this goals are known for pure scheduling problems, it is not so clear how resource assignments are best treated. This section discusses the main search methods adopted in CP for resource allocation and scheduling problems (Section 6.1) and in OR for the MRCPSP (Section 6.2), trying to outline similarities and point out the main strengths/weaknesses of each approach.

6.1 Search strategies in constraint programming

Exact CP algorithms for resource allocation and scheduling problems are usually based on Depth First Search. The nodes of the search tree represent partially instantiated schedules, where scheduled activities have fixed start time; time windows for the unscheduled activities are maintained via the domains of start/end variables and updated via propagation. Search proceeds by opening choice points and posting different branching constraints along each branch. Distinct strategies differ for the type of posted constraints and for the heuristics used to take non-deterministic decisions.

Two-stage search Here, two-stage search refers to any tree search method taking into account resource assignment and scheduling variables in successive, distinct phases. This is the default search method for alternative resources in IBM ILOG

Scheduler, where all resource assignment decisions are taken in a first stage, and a branching scheme for pure scheduling is then applied. The approach is very simple to implement, but tends to considerably under-exploit the joint propagation of precedence and resource constraints. The best results are obtained for project graphs featuring a very small number of arcs. The method is also considered for the Disjunctive Temporal Problem with Finite Domain constraints [64] (see Section 3.1.2).

"Left-justified random" and "next or successor-but-not-next" The Left-Justified Random strategy has been introduced in [67] for pure scheduling problems. The method finds the smallest earliest *finish* time of all the unscheduled activities and then identifies the set of activities which are able to *start* before this time. One of the activities in this set (say a_i) is selected randomly and scheduled at its earliest start time. When backtracking, the alternative commitment is to update the earliest start time of the activity to the minimum earliest finish time of all other activities on the same resource as a_i. The approach has been extended to models with alternative activities in [12], by preventing activities with ex_i variable bound to 0 from being selected for scheduling. Then, when the chosen activity is scheduled, it is simultaneously selected for execution. On backtrack, the earliest start time of the a_i is updated, but the corresponding execution variable is not modified (this ensures that search is complete).

By scheduling activities at their earliest start time, Left Justified Random focuses on the construction of active schedules (see Section 5.2.2). This considerably narrows the search space and incurs no loss of optimality in case of regular performance measures (very common in practice). In the experimental results in [12], Left-Justified Random reports quite poor results compared to Precedence Constraint Posting guided by texture based heuristics; we conjecture that changing the activity selection criterion may improve the outcome.

A closely related strategy is employed in [34] for an assignment and scheduling problem with unary resources and setup times. The method makes use of binary choice points. Let L be the set of the last activities scheduled on each resource; in the first branch, the activity a_i with minimum earliest start time is scheduled to be "next" after some activity a_j in L and assigned to the same resource. On backtracking, a_i is forced to be a "successor, but not next" and no resource assignment is performed. This strategy mainly differs from Left-Justified random for the activity selection criterion (which tends to be more effective) and for the lack of an earliest start time update on backtracking. The approach has some similarity with Precedence Constraint Posting, since it does not require to immediately assign a start time to the selected activity.

"Schedule or postpone" This strategy is proposed in [60] for pure scheduling problems. At each search node an activity a_i is selected for branching; typically, this is the one with the lowest $EST(a_i)$, while $LET(a_i)$ is used to break ties. Then a binary choice point is opened and a_i is scheduled at $EST(a_i)$ along the first branch. Upon backtracking the activity is marked as non-selectable (i.e. postponed) until its earliest start time is modified by propagation. In other words, the time window update performed on backtracking by Left-Justified Random is delegated here to resource constraint propagation, thus making the approach more general. Moreover, postponing activities reduces the size of the search space by preventing scheduling

choices to be repeated. The main underlying idea is once again to focus search on the production of active schedules. However, the method performs well even for many non-regular objectives, despite no optimality guarantee can be given in that case.

Intuitively, generalizing the schedule-or-postpone strategy requires to take into account resource assignment decisions and to focus on the production of tight (rather than active) schedules. This is attempted in [17] within an approach with alternative resources, by adding an assignment stage before opening a schedule-or-postpone choice point. The method works very well for graphs with many precedence constraints, since in this case interleaving assignment and scheduling decisions triggers better propagation compared to two-stage search. Note however that, in order to produce tight schedules, it would be sufficient to postpone the activity only once *all* the possible resource assignments have been tried. As a consequence, the proposed strategy explores unnecessary portions of the search space, leaving room for future improvements.

Precedence constraint posting (PCP) This search method has been designed for pure scheduling problems and proceeds by resolving possible resource conflicts through the addition of precedence constraints. The key idea is to identify minimal sets of activities causing a resource over-usage in case of overlapping execution, i.e. so called Minimal Critical Sets (MCS), introduced in [47] as "Minimal Forbidden Sets" and first used in CP in [22]. Due to the minimality property, the occurrence of a MCS can be prevented by adding a single precedence constraint between any two activities in the set (a so-called resolver). Branching is done either by enumerating all possible conflict resolution choices, or by opening a binary choice point where a selected resolver is alternatively posted or forbidden. MCS can be detected (1) by enumeration [54], (2) by sampling peaks over an earliest start schedule or over a worst case resource usage envelope [71], or (3) by solving a minimum flow problem [62].

This search method has been applied to resource assignment and scheduling problems in [12], modeled via alternative activities. MCSs are identified and solved by either posting a precedence constraint or by setting to 0 the execution variable ex_i of some activity. Search is performed via binary choice points. The PCP approach allows to resolve conflicts in a non-chronological order (e.g. the hardest first). As a side effect, the resulting schedules are not necessarily tight (see the example in [82]) and left-shift rules are difficult to apply.

The actual conflict and the resolver used for branching are chosen according to some heuristic. In [12], three different heuristics are proposed and evaluated: the first one is an adaptation of a slack based heuristic from the literature. The latter two are texture based heuristics, corresponding in practice to conflict and resolver selection procedures: the main underlying idea is to rely on some information extracted from the current problem state (a so-called texture) to identify a critical resource r_k^* and a critical time point t^*. Then sequencing decisions are taken on the activities with a chance of overlap and cause a conflict at t^*. In detail, the paper considers two textures: (1) a probabilistic resource usage profile and (2) a conflict probability function over time, from which the critical resource and time point are extracted. Both approaches perform very well in the experimental results. As a main difference with classical PCP conflict selection, the heuristics may identify critical (r_k^*, t^*) even if no conflict arises for that pair and should be therefore coupled with a conflict

detection procedure. Note the worst case usage envelope employed in [71] to detect MCS can be considered a form of texture. The ideas exposed in this section are very closely related to Minimal Forbidden Set branching for the MRCPSP (see Section 6.2).

6.2 Branching schemes for the MRCPSP

All the exact approaches developed in an OR context for the MRCPSP are based on tree search. Unlike in Constraint Programming, where a search node always corresponds to a state of the domain store, branching schemes from the Operations Research literature rely on different types of schedule representation.

Precedence tree branching This branching strategy is introduced in [84], but received a major improvement by Patterson in [68]. Each node of the search tree corresponds to a resource- and precedence-feasible *partial schedule*, i.e. a schedule of a set S of activities, built in chronological order from time 0. No updated information (e.g. range of possible start times) is maintained for unscheduled activities.

The strategy consists of scheduling at each step of the search tree an activity whose predecessors have all been scheduled. Therefore, for each search node with partial schedule S a set of eligible activities $E(S)$ is unambiguously defined. On backtrack, a different activity is chosen, so that each path from the root of the search tree to the leaves corresponds to a possible linearization of the partial order induced on A by the precedence graph. A *mode alternative* within this branching scheme is an assignment of a mode μ_h to the target activity a_i, which is performed after the scheduling decision. On backtrack, different modes are tested, so that a scheduling decision on a_i is only undone once all modes for a_i are tested.

The algorithm by Patterson has been further improved in [81] and [79], in particular with the introduction of bounding rules. The structure of the approach is well described in [38]. The precedence tree method suffers from symmetry issues. In particular, if two activities a_i, a_j can be independently assigned the same start time, the method will always test both enumeration sequences. This is countered by the application of the single enumeration rule (see Section 5.2), which in fact provides the highest benefits on this branching scheme.

Mode and delay alternatives This branching method is introduced in [23, 29] for the RCPSP and is adapted to the multi-mode case in [80]; it is well described in [38] and recently used in [26]. Each node of the search tree is associated to a feasible *partial schedule* S and a time instant t. A clear distinction is then made between completed activities at time t (say $C(S, t)$) and activities in process (say $P(S, t)$); eligible activities for scheduling are those whose predecessors have all completed execution.

Then an attempt is made to schedule *all* eligible activities and they are added to the set of activities in process. Of course this may cause a conflict; in such a case the method branches by *withdrawing* from execution the so-called delay alternatives. Those are: (1) activities in process, i.e. in $P(S, t)$ and (2) activities such that, if they are removed, no resource conflict occurs. A delay alternative is called minimal if none of its proper subsets is a delay alternative. Branching on minimal delay alternatives is sufficient to explore the whole search space. If no resource conflict occurs, the only minimal delay alternative is the empty set.

This method differs from the precedence tree based one in two regards: (1) the process branches on *sets* of activities and (2) activities scheduled at a search node may be withdrawn from execution later on: in case this is not done, the algorithm only builds so called non-delay schedules [82] and may miss the optimal solution (even in case the objective is regular).

Activities are assigned a mode when they are first inserted in the $P(S, t)$ and they retain the mode assignment when they are withdrawn from execution. As a consequence, when the simultaneous execution of all eligible activities is probed, some activities already possess a mode, while the remaining ones are modeless. A mode alternative in this search strategy is a mapping that assigns a mode to every activity in the modeless eligible set. On backtracking, when the subtree corresponding to a delay alternative has been completely explored, a different mode alternative is picked.

Mode and extension alternatives This method was proposed in [83] for the RCPSP, while the adaptation to the multi-mode case is discussed in [38]. The approach is similar to mode and delay alternatives: each search node corresponds to a *partial schedule* S and a time instant t, for which sets $C(S, t)$ and $P(S, t)$ are identified. The activities with all predecessors in $C(s, t)$ form the *eligible set* $E(S, t)$.

The current partial schedule is extended by starting a *subset* of the eligible activities without violating the renewable resource constraints. Conversely, in delay alternatives all eligible activities are started, then some are withdrawn from execution. In order to ensure that the algorithm terminates, empty extension alternatives may be disregarded if $P(s, t) = \emptyset$ (i.e. no activity is in process). However, if there are currently activities in process, the empty set is always an extension alternative which must be tested in order to guarantee optimality. In case this is not done, the algorithm only builds non-delay schedules and may miss the optimal solution.

A *mode alternative* is a mapping of modes to activities and occurs as soon as they become *eligible*, before an extension alternative is selected. The backtracking mechanism is the same as for delay alternatives. This branching scheme is proven to be dominated by precedence tree and delay alternative branching in [38], at least when no bounding rule is applied.

Dichotomization and time window tightening A branching scheme based on dichotomization is proposed for the MRCPSP in [94]. The approach operates on a time indexed model and is based on Special Ordered Sets (SOS, [9]); in detail, each considered SOS includes all the binary variables $\mathtt{E_{i,h,t}}$ referring to a single activity.

Given a fractional LP solution, branching is performed by splitting the SOS referring to an activity a_i, based on its average finish time t_b. The first subset contains variables with $t \leq t_b$, the second with $t > t_b$. Two branches are defined by respectively setting the variables in each subset to zero. The search method is coupled with a bound-tightening step at each branch. The approach obtains interesting results.

A related approach for the Multi-skill Project Scheduling Problem (affine to the MRCPSP) is proposed by Pessan et al. in [69] and is based on time windows tightening. At each node a time window is selected and reduced until all activities have their starting point fixed. During the process two lower bounds are used: one based on the compatibility graph that checks which activities can be scheduled at the same time, and one based on energetic reasoning that checks the mandatory parts.

Minimal forbidden sets Minimal Forbidden Sets, known in CP as Minimal Critical Sets, are introduced in [47] in the early 80s; they can be used to define choice points in tree search as described in Section 6.1. Unlike in the CP literature where branching is mainly done via binary precedence constraints (resolvers), OR methods have explored the use of *disjunctive precedence constraints*. A disjunctive precedence constraint requires a specific activity a_i in the MCS to execute after *at least one* other activity a_j in the set: the specific a_j causing the delay may be left undecided until all start times are assigned. The method is devised in [76] for the RCPSP and is applied in [39] to the multi-mode case.

With this branching strategy, a search node corresponds to a *fictitious* schedule rather than to a partial schedule. A fictitious schedule assigns a set of possible modes and a provisional start time to each activity of the project. The resulting representation is very similar to the one obtained in CP via the domain store. Disjunctive precedence constraints are based on the same idea of delay alternatives, but enable one to consider conflicts in non-chronological order. In particular, the method described in [39] systematically branches on the (estimated) hardest decision. The specific branching rule applied depends on the type of this decision: if this is a mode decision, each branch is a mode assignment for the corresponding activity. If it is a conflict decision, each branch is a possible disjunctive precedence constraint to resolve the conflict.

The use of disjunctive precedence relation allows a remarkable reduction of the search tree compared to binary resolvers. As a drawback, with this type of precedence relations the feasible space becomes a *union* of convex polyhedra and Generalized Arc Consistency on the temporal constraints can no longer be achieved in polynomial time.

7 Decomposition-based solution methods

Most of the decomposition approaches for assignment and scheduling problems are strongly related to Logic based Benders Decomposition (LBD), introduced in [46] and formalized in [40, 45]. LBD generalizes classic Benders Decomposition [15] and considerably broadens its application field, by replacing the linear dual used for cut generation with a so-called *inference dual*. The method breaks a combinatorial problem into a master and a subproblem, which are solved in sequence. The master solution is used to prime the subproblem, then a cut is generated (Benders cut) and permanently added to the master problem. The process repeats until the master and the sub-problem converge in value and optimality is proved. In the context of allocation and scheduling, resource assignment variables x_{ik} are typically included in the master problem, while scheduling variables s_i, e_i only appear in the subproblem. Assuming a minimization objective, the cost function $F(\overline{x}, \overline{s}, \overline{e})$ is replaced by a relaxed function $G(\overline{x})$ such that $G(\overline{x}) \leq F(\overline{x}, \overline{s}, \overline{e})$. The approach allows one to use heterogeneous solution techniques: quite often MILP is employed with the master and CP with the subproblem, due to the effectiveness of CP constraints and search strategies for pure scheduling.

Benders cuts at each iteration, once an assignment of master problem variables $\overline{x}'$ is available, are obtained from the solution of an inference dual. In detail, let the function $F^*(\overline{x})$ denote the minimum value of $F(\overline{x}, \overline{s}, \overline{e})$ corresponding to an assignment

of the master problem variables: the inference dual consists in computing the value $F^*(\overline{x}')$. Then, the algorithm designer is responsible to identify a bounding function $\beta_{\overline{x}'}(\overline{x})$ such that $\beta_{\overline{x}'}(\overline{x}) \leq F^*(\overline{x})$ and $\beta_{\overline{x}'}(\overline{x}') = F^*(\overline{x}')$. In other words, $\beta_{\overline{x}'}(\overline{x})$ equals the optimal $F(\overline{x}, \overline{s}, \overline{e})$ when $\overline{x} = \overline{x}'$, otherwise the function provides a valid bound. The subscript of the bounding function denotes the $\overline{x}$ values used for its construction. The Benders cut consists in forcing the master-problem objective to be greater then the bounding function, i.e. $F(\overline{x}) \geq \beta_{\overline{x}'}(\overline{x})$. At each iteration, the cumulated Benders cuts prevent old master problem solutions from being re-proposed.

The approach was first applied to allocation and scheduling problems in [37, 48], obtaining dramatic improvements over pure CP/MILP methods and over a hybrid algorithm using CP for branching and LP-based lower bounds. Other application examples include [42] for tardiness costs, [21] for scheduling on hard-real time systems, [73] for power consumption minimization on multi-core CPUs.

In case of resource based objectives (see Section 2.1.3) the sub-problem has no cost function, making CP an even better candidate for its solution; this is well documented in [44, 48]. In that case, Benders cuts can be simply formulated as constraints on the x_{ik} variables rather than bounds on the master cost function. Sometimes, it is possible to further decouple the subproblem into independent components (e.g. a scheduling problem for each resource), with strong efficiency improvements: this typically occurs when no precedence constraint is specified and is discussed in Section 3.3. In general, a tighter bounding function $\beta_{\overline{x}'}(\overline{x})$ results in quicker convergence, providing motivation for cut strengthening techniques: an efficient ad hoc method is reported in [41].

Balancing Quite often, the master problem turns out to be considerably harder to solve than the sub-problem, therefore limiting the number of full iterations performed in a given amount of time. This is a relevant issue and can be dealt with in a number of ways:

- by not solving the master problem to optimality: this especially makes sense in case of scheduling dependent objectives, where optimal values for the *relaxed* cost function $F(\overline{x})$ do not necessarily result in better solutions. This observation is reported in [85] and provides some arguments for the use of CP in the master problem;
- by solving the sub-problem while still branching to search for a master solution: this is the key idea behind so-called Branch-and-Check [85] and requires a method to formulate the sub-problem when some master problem variables are unassigned—e.g. because the master and the sub-problem may share some variables;
- By using stronger and more computation expensive cuts. For example, improved cuts can be obtained via explanation minimization by repeatedly solving polynomial complexity sub-problems [21] or even NP-hard ones [63].
- by applying LBD recursively on the master problem itself, obtaining a multi-stage decomposition: [17] describes the approach in a specific application settings and provides empirical rules to balance the computational load of the different decomposition stages.

Subproblem relaxation Logic based Benders Decomposition incurs the risk of losing valuable information due to decoupling; addressing this issue is the fundamental

problem during the design of a LBD approach: this is of course tackled by devising effective cuts, but the interaction between the two stages can be further tightened by introducing in the master problem a so-called *sub-problem relaxation*. This may consist of additional constraints (i.e. some of the subproblem constraints, possibly relaxed) or may be an additional bounding function on the master problem objective. The actual formulation has to be given case by case: examples can be found in basically all LBD related references in this paper. The use of an effective subproblem relaxation often has a tremendous impact on the performance: in [85] it is shown how removing the relaxation from the model in [48] increases the search time by several orders of magnitude. Finally, in case of cost functions depending on both assignment and scheduling decisions, a subproblem relaxation guides the master problem solver towards better solutions.

8 Core solution ideas

Despite their number and variety, all the presented techniques tend to cluster around some core strategies to cope with the specificities of Assignment and Scheduling problems. In the following, we briefly discuss what we believe are the three main ones.

8.1 Improving propagation between assignment and scheduling variables

Assignment and Scheduling problems are particularly challenging for CP approaches, since classical scheduling constraints tend to propagate quite poorly until many assignment decisions are taken. The need to deal with this critical issue has lead CP researchers to devise novel modeling approaches. Those are designed to be sufficiently general to describe practical problems and yet sufficiently restricted to provide useful information to propagation algorithms.

Alternative resources are one such example: they can only model very simple assignment decisions, but enable stronger filtering via multiple fake activities, redundant resources and unit propagation on the assignment variables (Section 4). Alternative activities provide similar advantages, but manage to model also multiple modes, complex assignment constraints and alternative process plans. Conditional Time Intervals are probably the most mature outcome of this research line: they have a clean formulation, provide the same benefits as alternative activities (thanks to explicitly stated alternative blocks) and support a more traditional modeling style by allowing optional activities out of alternative blocks. Additionally, Time Interval Variables introduce novel forms of joint temporal and assignment filtering.

8.2 Exploiting problem specificities to focus search

This approach has been primarily pursued in the Operations Research, due to the lack of effective MILP bounds for scheduling problems. The most relevant outcome is a set of powerful search strategies (Section 6.2) and dominance rules (Section 5.2).

Disjunctive precedence constraints are a powerful tool to reduce the size of the search tree. Branching schemes based on minimal delay alternatives report remarkable results compared to other OR approaches. As far as dominance

rules are concerned, the best results are obtained by restricting search to tight and mode minimal schedules and by removing permutation symmetries at search time.

Finally, despite the fact that bounds based on linear relaxations tend to be weak, a linear model may still provide a guide for search strategies. This is particularly valuable for cost functions with poor back-propagation, such as tardiness and earliness based objectives: in this case, a linear relaxation may be used to identify optimal starting times (see Section 5.1). This method is exploited in CP in the context of self-adapting large neighborhood search [56].

8.3 Using decomposition to reduce the search time

Decomposition based approaches take a radically different perspective, moving the focus away from the issue of exploiting the connection between assignment and scheduling variables. Conversely, they rely on maximizing the benefits of decomposition: smaller and easier to solve subproblems, plus the ability to use heterogenous techniques. The most relevant examples are probably Logic based Benders Decomposition (LBD) and Branch-and-Check (Sections 3.3 and 7).

The main drawback in this case is the loss of propagation between assignment and scheduling decisions. Benders cuts are the basic mean to cope with this issue. Using basic no-goods does not in general compensate for the decoupling, but refinement techniques may greatly improve the cut effectiveness. Introducing a subproblem relaxation in the master is a second, very effective mean to reduce the adverse effects of decomposition.

Published results from the last 10 years (such as [44, 48]) are characterized by a quite sharp dominance of LBD like approaches over pure CP or MILP ones. For this reason, hybrid, decomposition based, approaches are suggested for solving to optimality most practical assignment and scheduling problems. The specific technique to be applied in the master and the subproblem should be chosen case by case. Using MILP or some strongly optimality-driven technique in the master problem is advised in case the cost function only depends on the master variables. Pure MILP and CP approaches can still provide very good results on specific scenarios: this is the case if the graph contains many precedence constraints and the assignment component is relatively simple (e.g. [17]), or with very tight resource constraints (e.g. [94]). Note also that pure CP approaches can be effectively employed in conjunction with decomposition based methods to quickly provide good quality solutions.

It is interesting to see how some of the most effective search methods in the OR literature (delay alternatives and disjunctive precedence constraints) have not been applied in a CP context so far. Integrating the two approaches provides an interesting improvement opportunity, but is not trivial since disjunctive precedence constraints limit temporal consistency and delay alternatives require to *withdraw* activities from execution. The effectiveness of the left-shift rule is well acknowledged in CP and the Schedule or Postpone strategy is an effective way to build active schedules when there are no assignment decisions. There is however a lack of CP methods designed to produce tight schedules for problems *with* assignment decisions. Similarly, search strategies for mode minimal schedules are largely uninvestigated in the CP community and may provide an interesting topic for future research. Finally, effective joint propagation on assignment and scheduling variables appears to be

inherently difficult for classical CP filtering. This provides a stimulating challenge for novel techniques such as Lazy Clause Generation or propagation based on Multi-valued Decision Diagrams.

9 Conclusions

We provided an overview of state of the art approaches for a class of resource allocation and scheduling problems, arising in a variety of real world settings. Given the number of problem variants we chose to focus this work on techniques to address individual problem traits, rather than on devising an exhaustive (most likely too complex) classification. In particular, we mainly drew the presented pool of algorithms and methods from scheduling related OR and CP literature. Constraint Programming is a natural candidate to support the integration of heterogeneous techniques: its typical distinction between model, propagation and search provided the backbone for the work organization. Hybrid methods were given prominent importance, as they proved to be particularly effective on allocation and scheduling problems. The whole paper can be considered as an attempt to provide hints for the development of new hybrid algorithms and novel filtering methods.

We decided to limit our discussion to exact approaches. However, given the impressive complexity of this class of problems, a very large number of works from the literature adopts heuristic solution methods. We hope our effort will provide motivation to the research community for surveying the state of the art of heuristics for resource assignment and scheduling.

References

1. Baptiste, P., Laborie, P., Le Pape, C., & Nuijten, W. (2006). Constraint-based scheduling and planning. *Foundations of Artificial Intelligence, 2*, 761–799.
2. Baptiste, P., & Le Pape, C. (1996). Disjunctive constraints for manufacturing scheduling: Principles and extensions. *International Journal of Computer Integrated Manufacturing, 9*(4), 306–310.
3. Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based scheduling*. Kluwer Academic Publishers.
4. Barták, R., & Cepek, O. (2008). Nested precedence networks with alternatives: Recognition, tractability, and models. In *Proc. of AIMSA* (pp. 235–246).
5. Barták, R., Cepek, O., & Surynek, P. (2007). Modelling alternatives in temporal networks. In *Proc. of IEEE SCIS* (pp. 129–136).
6. Barták, R., Čepek, O., & Hejna, M. (2008). Temporal reasoning in nested temporal networks with alternatives. *Recent Advances in Constraints, LNCS, 5129*, 17–31.
7. Barták, R., Čepek, O., & Surynek, P. (2008). Discovering implied constraints in precedence graphs with alternatives. *Annals of Operations Research, 180*(1):233–263.
8. Bartusch, M., Möhring, R. H., & Radermacher, F. J. (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research, 16*(1), 199–240.
9. Beale, E. M. L., & Forrest, J. J. H. (1976). Global optimization using special ordered sets. *Mathematical Programming, 10*(1), 52–69.
10. Beasley, J. E., & Krishnamoorthy, M. (2000). Scheduling aircraft landings-the static case. *Transportation Science, 34*(2), 180–197.
11. Beck, J. C., & Fox, M. S. (1999). Scheduling alternative activities. In *Proc. of AAAI/IAAI* (pp. 680–687).
12. Beck, J. C., & Fox, M. S. (2000). Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence, 121*(1–2), 211–250.

13. Beldiceanu, N., Carlsson, M., Demassey, S., & Poder, E. (2011). New filtering for the cumulative constraint in the context of non-overlapping rectangles. *Annals of Operations Research, 184*(1), 27–50.
14. Bellenguez-Morineau, O., & Néron, E. (2007). A Branch-and-Bound method for solving multi-skill project scheduling problem. *RAIRO - Operations Research, 41*(02), 155–170.
15. Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik, 4*(1), 238–252.
16. Benini, L., Bertozzi, D., Guerri, A., & Milano, M. (2005). Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Proc. of CP* (pp. 107–121).
17. Benini, L., Lombardi, M., Milano, M., & Ruggiero, M. (2011). Optimal allocation and scheduling for the cell BE platform. *Annals of Operations Research, 184*(1), 51–77.
18. Brucker, P., Drexl, A., Möhring, R. H., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research, 112*(1), 3–41.
19. Buddhakulsomsiri, J., & Kim, D. (2006). Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. *European Journal of Operational Research, 175*(1), 279–295.
20. Burkard, R. E., Dell'Amico, M., & Martello, S. (2009). *Assignment problems*. Society for Industrial Mathematics.
21. Cambazard, H., Hladik, P. E., Déplanche, A. M., Jussien, N., & Trinquet, Y. (2004). Decomposition and learning for a hard real time task allocation problem. In *Proc. of CP* (pp. 153–167).
22. Cesta, A., Oddi, A., & Smith, S. F. (1998). Scheduling multi-capacitated resources under complex temporal constraints. In *Proc. of CP* (pp. 465–465).
23. Christofides, N., Alvarez-Valdes, R., & Tamarit, J. M. (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research, 29*(3), 262–273.
24. Coban, E., & Hooker, J. N. (2010). Single-facility scheduling over long time horizons by logic-based benders decomposition. In *Proc. of CPAIOR* (pp. 87–91).
25. De Reyck, B., Demeulemeester, E., & Herroelen, W. (1998). Local search methods for the discrete time/resource trade-off problem in project networks. *Naval Research Logistics, 45*(6), 553–578.
26. Deblaere, F., Demeulemeester, E., & Herroelen, W. (2010). Reactive scheduling in the multi-mode RCPSP. *Computers & Operations Research, 38*(1), 1–12.
27. Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence, 49*(1–3), 61–95.
28. Demeulemeester, E., De Reyck, B., & Herroelen, W. (2000). The discrete time/resource trade-off problem in project networks: A branch-and-bound approach. *IIE Transactions, 32*(11), 1059–1069.
29. Demeulemeester, E. L., & Herroelen, W. (1992). A branch-and-bound procedure for multiple resource-constrained project scheduling problem. *Management Science, 38*(12), 1803–1818.
30. Drexl, A., & Gruenewald, J. (1993). Nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions, 25*(5), 74–81.
31. Elmaghraby, S. E. (1977). *Activity networks: Project planning and control by network models*. New York: Wiley.
32. Elmaghraby, S. E., & Kamburowski, J. (1992). The analysis of activity networks under generalized precedence relations (GPRs). *Management Science, 38*(9), 1245–1263.
33. Ernst, A.T., Krishnamoorthy, M., & Storer, R.H. (1999). Heuristic and exact algorithms for scheduling aircraft landings. *Networks, 34*(3), 229–241.
34. Focacci, F., Laborie, P., & Nuijten, W. (2000). Solving scheduling problems with setup times and alternative resources. In *Proc. of ICAPS*.
35. Focacci, F., Lodi, A., & Milano, M. (1999). Cost-based domain filtering. In *Proc. of CP* (pp. 189–203).
36. Gambardella, L. M., & Mastrolilli, M. (1996). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling, 3*, 3.
37. Harjunkoski, I., Jain, V., & Grossman, I. E. (2000). Hybrid mixed-integer/constraint logic programming strategies for solving scheduling and combinatorial optimization problems. *Computers & Chemical Engineering, 24*(2–7), 337–343.
38. Hartmann, S., & Drexl, A. (1998). Project scheduling with multiple modes: A comparison of exact algorithms. *Networks, 32*(4), 283–297.

39. Heilmann, R. (2003). A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research, 144*(2), 348–365.
40. Hooker, J. (2000). *Logic-based methods for optimization: Combining optimization and constraint satisfaction*. New York: Wiley.
41. Hooker, J. N. (2005). A hybrid method for planning and scheduling. *Constraints, 10*(4), 385–401.
42. Hooker, J. N. (2005). Planning and scheduling to minimize tardiness. In *Proc. of CP* (Vol. 3709, pp. 314–327).
43. Hooker, J. N. (2006). An integrated method for planning and scheduling to minimize trdiness. *Constraints, 11*(2–3), 139–157.
44. Hooker, J. N. (2007). Planning and scheduling by logic-based benders decomposition. *Operations Research, 55*(3), 588.
45. Hooker, J. N., & Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming, 96*(1), 33–60.
46. Hooker, J. N., & Yan, H. (1995). Verifying logic circuits by Benders decomposition. In *Proc. of CP* (pp. 267–288).
47. Igelmund, G., & Radermacher, F. J. (1983). Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks, 13*(1), 1–28.
48. Jain, V., & Grossmann, I. E. (2001). Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing, 13*(4), 258–276.
49. Kelley, J.E., & Walker, M. R. (1959) Critical-path planning and scheduling. In *Proc. of eastern joint IRE-AIEE-ACM conference* (pp. 160–173). ACM: New York.
50. Kolisch, R. (1997). PSPLIB - A project scheduling problem library. *European Journal of Operational Research, 96*(1), 205–216.
51. Kolisch, R., & Drexl, A. (1997). Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions, 29*(11), 987–999.
52. Kwok, Y. K. (1996). Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *Parallel and Distributed Systems, IEEE, 7*(5), 506–521.
53. Kwok, Y. K., & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR), 31*(4), 406–471.
54. Laborie, P. (2005). Complete MCS-based search: Application to resource constrained project scheduling. In *Proc. of IJCAI* (pp. 181–186). Professional Book Center.
55. Laborie, P. (2009). IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In *Proc. of CPAIOR* (pp. 148–162).
56. Laborie, P., & Godard, D. (2007). Self-adapting large neighborhood search: Application to single-mode scheduling problems. In *Proc. of MISTA*.
57. Laborie, P., & Rogerie, J. (2008). Reasoning with conditional time-intervals. In *Proc. of FLAIRS* (pp. 555–560).
58. Laborie, P., Rogerie, J., Shaw, P., & Vilìm, P. (2009). Reasoning with conditional time-intervals part II: An algebraical model for resources. In *Proc. of FLAIRS* (pp. 201–206).
59. Le Pape, C. (1994). Using a constraint-based scheduling library to solve a specific scheduling problem. In *Proc. of AAAI-SIGMAN*.
60. Le Pape, C., Couronné, P., Vergamini, D., & Gosselin, V. (1994). Time-versus-capacity compromises in project scheduling. In *Proc. of PLANSIG*.
61. Leupers, R. (2000). Instruction scheduling for clustered VLIW DSPs. In *Proc. of PACT* (pp. 291–).
62. Lombardi, M., & Milano, M. (2009). A precedence constraint posting approach for the RCPSP with time lags and variable durations. In *Proc. of CP* (pp. 569–583).
63. Lombardi, M., & Milano, M. (2010). Allocation and scheduling of conditional task graphs. *Artificial Intelligence, 174*(7–8), 500–529.
64. Moffitt, M. D., Peintner, B., & Pollack, M. E. (2005). Augmenting disjunctive temporal problems with finite-domain constraints. In *Proc. of AAAI* (pp. 1–6).
65. Nuijten, W. (1994). *Time and resource constrained scheduling: A constraint satisfaction approach*. PhD thesis, Technische Universiteit Eindhoven.
66. Nuijten, W., Bousonville, T., Focacci, F., Godard, D., & Le Pape, C. (2004). Towards an industrial manufacturing scheduling problem and test bed. In *Proc. of PMS*.
67. Nuijten, W. P. M., Aarts, E. H. L., van Arp Talmaan Kip, D.A.A., & van Hee, K.M. (1993). Randomized constraint satisfaction for job shop scheduling. In *Proc. of IJCAI* (pp. 251–262). Chambery, France.

68. Patterson, J.H., Slowinski, R., Talbot, F.B., & Weglarz, J. (1989). An algorithm for a general class of precedence and resource constrained scheduling problems. In R. Slowinski & J. Weglarz (Eds.), *Advances in Project Scheduling* (Part I, Chapter I, pp. 3–28). Amsterdam: Elsevier Science Publishers.
69. Pessan, C., Bellenguez-Morineau, O., & Néron, E. (2007). Multi-skill project scheduling problem and total productive maintenance. In *Proceedings of MISTA* (pp. 608–610).
70. Poder, E., & Beldiceanu, N. (2008). Filtering for a continuous multi-resources cumulative constraint with resource consumption and production. In *Proc. of ICAPS* (pp. 264–271).
71. Policella, N., Cesta, A., Oddi, A., & Smith, S. F. (2007). From precedence constraint posting to partial order schedules: A CSP approach to robust scheduling. *AI Communications, 20*(3), 163–180.
72. Radermacher, F. J. (1985). Scheduling of project networks. *Annals of Operations Research, 4*(1), 227–252.
73. Ruggiero, M., Pari, G., Guerri, A., Benini, L., Milano, M., Bertozzi, D., et al. (2007). A cooperative, accurate solving framework for optimal allocation, scheduling and frequency selection on energy-efficient mpsocs. In *Proc. of SoC* (pp. 1–4). IEEE.
74. Sabzehparvar, M., & Seyed-Hosseini, S. M. (2007). A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. *The Journal of Supercomputing, 44*(3), 257–273.
75. Schutt, A., Feydy, T., Stuckey, P.J., & Wallace, M. (2009). Why cumulative decomposition is not as bad as it sounds. In *Proc. of CP* (pp. 746–761).
76. Schwindt, C. (1998). *Verfahren zur Lösung des ressourcenbeschränkten Projektdauerminimierungsproblems mit planungsabhängigen Zeitfenstern*. Shaker.
77. Smith, S. F., Cheng, C. C. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proc. of AAAI* (pp. 139–139).
78. Kedad-Sidhoum, S., Solis, Y. R., & Sourd, F. (2008). Lower bounds for the earliness–tardiness scheduling problem on parallel machines with distinct due dates. *European Journal of Operations Research, 189*(3), 1305–1316.
79. Sprecher, A., & Drexl, A. (1998). Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research, 107*(2), 431–450.
80. Sprecher, A., Hartmann, S., & Drexl, A. (1997). An exact algorithm for project scheduling with multiple modes. *OR Spectrum, 19*(3), 195–203.
81. Sprecher, A., & Hwang, C. L. (1994). *Resource-constrained project scheduling: Exact methods for the multi-mode case*. New York: Springer.
82. Sprecher, A., Kolisch, R., & Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research, 80*(1), 94–102.
83. Stinson, J. P., Davis, E. W., & Khumawala, B. M. (1978). Multiple resource-constrained scheduling using branch and bound. *IIE Transactions, 10*(3), 252–259.
84. Talbot, F. B. (1982). Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science, 28*(10), 1197–1210.
85. Thorsteinsson, E. S. (2001). Branch and check: A hybrid framework integrating mixed integer programming and constraint programming. In *Proc. of CP* (pp. 16–30). Paphos, Cyprus.
86. Timpe, C. (2002). Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum, 24*(4), 431–448.
87. Toth, P., & Vigo, D. (2002). *The vehicle routing problem* (Vol. 9). Society for Industrial Mathematics.
88. Van Peteghem, V., & Vanhoucke, M. (2010). A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research, 201*(2), 409–418.
89. Vilím, P. (2009). Max energy filtering algorithm for discrete cumulative resources. In *Proc. of CPAIOR* (pp. 294–308).
90. Vilím, P. (2011). Timetable edge finding filtering algorithm for discrete cumulative resources. In *Proc. of CPAIOR* (pp. 230–245).
91. Vilím, P., Barták, R., & Cepek, O. (2005). Extension of (n log n) filtering algorithms for the unary resource constraint to optional activities. *Constraints, 10*(4), 403–425.

92. Vo, S., & Witt, A. (2007). Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. *International Journal of Production Economics, 105*(2), 445–458.
93. Zapata, J. C., Hodge, B. M., & Reklaitis, G. V. (2008). The multimode resource constrained multiproject scheduling problem: Alternative formulations. *AIChE Journal, 54*(8), 2101–2119.
94. Zhu, G., Bard, J. F., & Yu, G. (2006). A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *INFORMS Journal on Computing, 18*(3), 377–390.