Journal of Scheduling (2022) 25:675–704
https://doi.org/10.1007/s10951-022-00750-w

Packing-based branch-and-bound for discrete malleable task scheduling

Roland Braune[1]

Accepted: 25 July 2022 / Published online: 22 September 2022

Abstract
This paper addresses the problem of scheduling chain-like structures of tasks on a single multiprocessor resource. In fact, sub-tasks of unit-time length and predefined size are aggregated to composite tasks that have to be scheduled without preemption, but subject to flexibility concerning resource allocation. This setting most closely resembles the problem of malleable task scheduling, with sub-tasks being the smallest atomic unit of allocation. The specific type of malleability is realized using precedence constraints with minimum and maximum time lags. A bin packing model is established for this scheduling problem and a corresponding, dedicated branch-and-bound algorithm is devised, alongside problem-specific bound tightening, symmetry breaking and dominance concepts. The efficacy of the solution approach is demonstrated based on extensive computational experiments, including randomized instances, adapted benchmark instances from the literature, and a small real-world data set. In comparison to mixed-integer and constraint programming formulations, the new method is able to achieve a considerably higher percentage of optimal solutions at computation times that are up to orders of magnitude smaller.

Keywords Multiprocessor scheduling · Malleable tasks · Precedence constraints · Bin packing · Branch-and-bound

1 Introduction

The relevance of parallel tasks (Drozdowski, 2009) goes beyond the initial context of multiprocessor scheduling rooted in information technology environments. In fact, tasks or activities that require more than one processor or, more generally, resource unit at a time are also typically found in manufacturing or project scheduling environments. The background of the research presented in this paper is a petrochemical research and development facility, conducting product tests on different kinds of resources. The resource capacities are aggregated to daily or weekly buckets, rendering the problem a multi-capacitated one. Each product test involves a number of test specimen, each having the same resource requirements. There is no upper limit on the number of specimen of the same product processed per day or week, but once the product test is started, gaps in its processing time window, that is, days or weeks where no specimen of this product are processed, are to be avoided. One of the objectives in the associated scheduling problem is to keep the utilization as high as possible in earlier time periods, while it is allowed to drop toward the end of the planning horizon. This requirement originates from the rolling-horizon nature of planning, aiming at both freezing parts of the schedule and leaving space for new tests. The analogy of this kind of problem to a multiprocessor scheduling problem with a total weighted completion time objective is pointed out in Braune (2019).

The subject of this paper is an abstract, purified version of the real-world setting outlined above. Test specimen are mapped to so-called task *slices*, each of which has unit time length and requires a discrete amount of resource units. They are linked through precedence constraints with predefined time lags. Minimum delays impose an order on the task slices, arranging them according to a chain-like structure, but they still permit more than one slice of a task to be executed at the same time. Maximum time lags, on the other hand, prevent high-level tasks from being interrupted.

As a consequence, the problem can be seen from two different perspectives. The first, low-level perspective is the one

✉ Roland Braune
roland.braune@univie.ac.at

[1] Department of Business Decisions and Analytics, Josef Ressel Center for Adaptive Optimization in Dynamic Environments, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

of the task slices. Related work in the corresponding literature is concerned with chains of rigid unit-time tasks (see, e.g., Blazewicz & Liu, 1996). However, those chains do not allow to simultaneously process two tasks of the same chain and also do not account for maximum delays. The second point of view emanates from the high-level tasks. Treating them as a compound during scheduling may result in varying resource requirements over time due to the potential concurrent processing of two or more of its slices. Modifying a task's resource requirements at scheduling time is known as *malleability* in the associated literature (see, e.g., Blazewicz et al., 2006). In the discrete case, the smallest increment or decrement is one single resource unit. For the problem at hand, in contrast, the smallest step size is a slice, leading to a somewhat more coarse-grained malleability. To the best of the author's knowledge, this kind of discrete malleable tasks has not been covered in the literature so far. No matter which viewpoint is adopted, the special type of constraints prohibit an immediate adaptation of existing solution approaches from this area of research.

The solution method proposed in this paper is based on a bin packing reformulation of the scheduling problem in question. The problem mapping itself arises from the unit-time slices of fixed size and the capacity limitation of the multiprocessor resource. Although existing work in this area in fact also covers precedence constraints, this specific combination of minimum and maximum delays has not been considered so far. The methodological contributions of this paper can be identified as follows:

- The maximum delays are exploited for the development of an innovative tightening approach for (existing) lower bounds on the objective function value.
- A symmetry breaking concept is presented, based on alternative, equivalent packing sequences.
- Several problem-specific dominance rules, based on the notion of feasible sets, are proposed.
- Construction heuristics known from standard bin packing are enhanced to properly cover the chain-like precedence constraints.
- A knapsack-based, precedence-aware heuristic proves very effective for the considered objective function.
- A subset sum-based, limited enumeration algorithm is used in a pre-optimization phase under particular problem configurations.

Finally, all the presented techniques are successfully embedded into a dedicated branch-and-bound (B&B) algorithm, as confirmed by an extensive computational study. To demonstrate the genericity of the proposed concepts and in particular the B&B algorithm, the experimental evaluation is extended toward the objective of minimizing the makespan or, equivalently, the number of allocated bins.

Section 2 introduces the scheduling problem that is the subject of this paper, both from a real-world and an abstract perspective. An overview of related scientific literature is given in Sect. 3. Section 4 introduces the packing-oriented view on the problem that is the basis for the methodological and computational part of the paper. The bound tightening technique is presented in Sect. 5, while the proposal of symmetry breaking and dominance rules is the subject of Sect. 6. A custom B&B algorithm as the central solution approach is outlined in Sect. 7. Computational results and a performance assessment of the proposed B&B approach are the subject of Sect. 8. Concluding remarks and an outlook on potential future research topics are given in the final Sect. 9.

2 Problem background and modeling

The motivation for the research work presented in this paper stems from a real-world project that the author conducted in cooperation with a petrochemical research laboratory a few years ago. The laboratory performs different types of tests for the product development department and the goal was to establish a decision support tool for an automated scheduling of the associated activities. The schedules should be rebuilt on a weekly basis to take into account newly added tests and also deviations from a previously established timeline. This *rolling horizon* principle is utilization oriented and gave rise to the specific objective function addressed in this paper.

The tests themselves can be of mechanical, thermic, or purely analytical (e.g., gas chromatography) nature. The objects on which the tests are executed are called *test specimen* and are in fact small pieces of plastic. A product test is usually made up of several individual tests, each involving multiple such specimen. A test is thus represented by an activity or task, and each task belongs to a job or work order. The task itself is structured into sub-tasks, each of which represents a test of one single specimen.

The execution of a test usually requires both a machine resource and a human resource (operator). Both kinds of resources are considered in an aggregate fashion with respect to time, meaning that resource capacities are given in terms of day-wise or weekly buckets. The capacity itself is subject to a discrete scale and measured in minutes. Human resources undergo an additional aggregation step: the capacity allocation is done on a *workgroup* level, reducing detail and complexity. The mission of the project was to develop a tailored, finite capacity scheduling approach for the test laboratory. The workgroup/operator resources are the scarcest and thus are of primary interest to the firm. The machines, on the other hand, are dedicated to specific tests, often in multiple versions, and therefore, the amount of resource contention is low in most cases. Therefore, the focus will be on resources of type workgroup in the following.

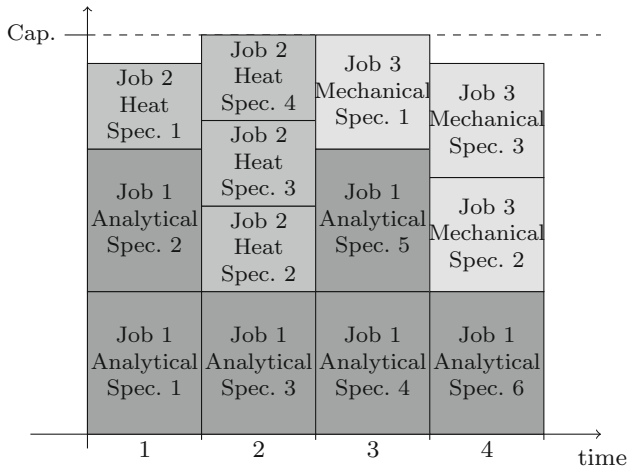

Fig. 1 Finite capacity scheduling of product tests on a bucket resource

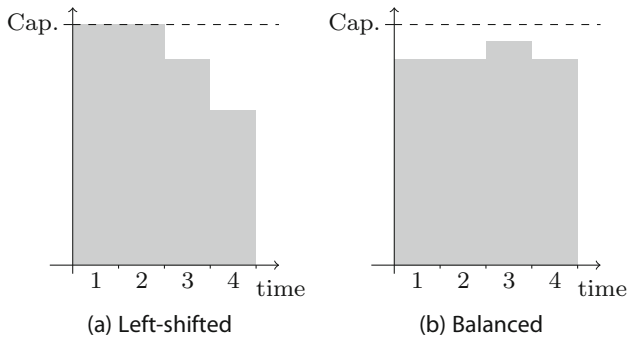

Fig. 2 Different principles of resource loading

When scheduling test activities on a resource, it is mandatory to respect custom precedence constraints between the sub-tasks. Figure 1 shows three such activities, belonging to different jobs, and each of them involves multiple test specimen. From a workflow management perspective, it is of crucial importance not to interrupt a test activity. Consider the analytical test in job 1. It consists of six individual tests, one for each specimen. While it is allowed to process more than one specimen in a time period, it is prohibitive to introduce breaks into the test flow. For example, it would not be possible to stop job 1 after time period 1 and continue its execution in time period 3, without any of its sub-tasks being scheduled in period 2. In other words, once started, at least one specimen of a test has to be processed in each time period of its execution time window.

From the company's point of view, the primary optimization objective is to keep the resource allocation profiles as *left-shifted* as possible. This in turn means that ideally no resource idle time exists in earlier time periods. Figure 2 shows such a left-shifted load profile, as opposed to a balanced one. The total load is the same in both cases, but the profile on the right-hand side (Fig. 2b) is not desirable. As already mentioned, schedules are generated on a rolling-horizon basis, usually for six weeks in advance. The management wants to "freeze" the schedule and to keep the resource utilization high for the very near future, that is, the immediately upcoming time periods. If the resource load is balanced and idle time exists in those time periods, new activities might be inserted at the corresponding positions during the next planning iteration, leading to disturbances and disruptions in the laboratory. A left-shifted schedule on the other hand prevents such disruptions and maximizes the resource utilization typically for the next two or three weeks.

When adopting a more formal perspective, the above real-world scheduling problem can be specified as a multiprocessor scheduling problem with particular precedence constraints. A set of n tasks (= test activities) $T = \{1, \ldots, n\}$ is to be processed on a set of m identical parallel processors, representing the daily (or weekly) capacity of a workgroup resource. Every task $i \in T$ consists of n_i sub-tasks, called *slices*, each of which corresponds to a test specimen. A slice is identified by a pair (i, j), with $i \in T$ and $j \in \{1, \ldots, n_i\}$. The number of processors to be allocated simultaneously for the execution of a task slice is denoted by $size_{i,j}$. All slices that make up a task $i \in T$ have unit processing time, that is, $p_{ij} = 1$, $j \in \{1, \ldots, n_i\}$, and share the same size, hence, $size_{i,1} = size_{i,2} = \ldots = size_{i,n_i}$.

A task with multiple slices is modeled as a *chain-like* structure by defining appropriate precedence constraints. Those constraints are imposed on every two slices that belong to one and the same high-level task and have adjacent slice indices. While minimum time lags ensure a partial order among slices, maximum time lags prevent the encompassing task from being preempted. Given a task $i \in T$, let $l_{i,j,j'}$ denote the start-to-start time lag (delay) between two task slices (i, j) and (i, j'), with $(j, j') \in \{1, \ldots, n_i\} \times \{1, \ldots, n_i\}$ and $j \neq j'$. Then, $l_{i,j,j+1} = 0$, for $j \in \{1, \ldots, n_i - 1\}$ and $l_{i,j,j-1} = -1$, for $j \in \{2, \ldots, n_i\}$.

A feasible schedule S is defined as a matrix of (integer) starting times $(s_{i,j})_{(i,j) \in T \times \{1,\ldots,n_i\}}$ that have to satisfy the following constraints:

$$s_{i,j+1} \geq s_{i,j} + l_{i,j,j+1}, \quad \forall i \in T, \ \forall j \in \{1, \ldots, n_i - 1\}, \tag{1}$$

$$s_{i,j+1} \leq s_{i,j} - l_{i,j+1,j}, \quad \forall i \in T, \ \forall j \in \{1, \ldots, n_i - 1\}. \tag{2}$$

Although maximum time lags are not present in classical chain precedence constraints known from scheduling (Blazewicz et al., 1996), the high-level tasks are referred to as *chains* henceforth, for ease of writing. Due to the fact that the number of scheduled slices of a chain may change from one time period to the next, the corresponding high-level task are considered *malleable*. Since resource allocation occurs on a discrete scale, this property is referred to as *discrete malleability* throughout the paper.

As far as the objective function is concerned, a left-shifted resource allocation profile, like the one shown in Fig. 2a, can be achieved by minimizing the total weighted completion time (TWCT) of the slices, where the weight w_{ij} of a slice is set equal to its size $size_{ij}$. The completion time $C_{i,j}$ of a slice in a feasible schedule S is equal to $s_{i,j} + 1$. To simplify the notation, and since all slices of a chain share the same weight, weights are considered at the chain level, using w_i as the common, individual weight for all slices of a chain $i \in T$. The value w_i is therefore also called the *base weight* of chain i in this specific context.

In the common three-field notation of Graham et al. (1979), the problem under consideration is of type $P \mid chains(l); size_{ij}; p_{ij} = 1; w_{ij} = size_{ij} \mid \sum w_{ij}C_{ij}$. The property $chains(l)$ means that the chain precedence constraints are subject to constant delays. The symbol l denotes the constant time lag that is imposed between two consecutive elements of the chain. As far as the problem's complexity is concerned, it can be considered a generalization of $P \mid size_i; p_i = 1 \mid \sum C_i$, that is, the multiprocessor scheduling problem with unit time tasks and total completion time objective. The latter problem has been proven to be NP-hard in the strong sense by Drozdowski and Dell'Olmo (2000), therefore also implying strong NP-hardness for the discrete malleable task scheduling problem at hand.

3 Related work

The sub-tasks or slices that are the core of the considered scheduling problem are classified as *rigid* parallel tasks according to the taxonomy of Drozdowski (2009). They require a fixed number of processors (or resource units) that cannot be changed during processing. Contrary to that, the high-level tasks (chains) can be seen as a special kind of *malleable tasks*. These are parallel tasks that are permitted to change the number of processors allocated to them during execution. Discrete and continuous variants of the problem are discussed by Blazewicz et al. (2004) and Blazewicz et al. (2006) under the objective of makespan minimization. The authors focus on two versions of this problem, including continuously and discretely divisible resources, and propose an $O(n)$ algorithm for both in case convex processing speed functions. In the completion (flow) time minimization context, Caramia and Drozdowski (2006) incorporate release times and deadlines and present two polynomially solvable cases, while the NP-hardness result of $P \mid pmtn; r_i \mid \sum C_i$ transfers to the general problem. Flow time minimization is also the subject of Hendel et al. (2015) who analyze this objective in a semi-malleable setting and on two processors only, resulting in a polynomial time algorithm for this variant.

The work of Sadykov (2012) is devoted to the total weighted completion time variant of the problem, denoted as $P \mid var; \delta_i \mid \sum w_i C_i$, where δ_i represents an upper bound on the number of allocatable processors. This makes it a generalization of $P \mid pmtn \mid \sum w_i C_i$, a problem that has been shown to be NP-hard in the strong sense. Zhang et al. (2013) directly prove that $Pm \mid var \mid \sum w_i C_i$ is strongly NP-hard as long as m is part of the problem input. They give a two-approximation algorithm for the general case and a polynomial time algorithm for problems with a restricted number of different task sizes (work contents). Wang et al. (2018) revisit this kind of problem and devise both, a polynomial time approximation scheme for a fixed number of machines and a polynomial time algorithm for a special case with fixed m and weights proportional to the task sizes.

Task malleability is also closely related to the field of *variable intensity* or *energy scheduling*. The latter was originally defined for discrete cumulative resources, representing the counterpart of multiprocessors in classical resource-constrained (project) scheduling (e.g., Baptiste et al., 1999). In the energy scheduling problem (Artigues et al., 2013), the duration of a task is not fixed, as the amount of resource units ("energy") allocated to it can vary during its execution, as long as it stays within predefined limits and a total required amount of energy is supplied. The problem setting has later been transferred also to the fully continuous case (Artigues & Lopez, 2015). Kis (2005) describes similar task characteristics, also in the context of continuously divisible renewable resources, referring to them as variable-intensity activities.

On the slice level, the problem covered in this paper is made up of unit time tasks of fixed size. Scheduling such tasks on parallel processors for makespan minimization ($P \mid size_i; p_i = 1 \mid C_{\max}$) dates back to the 1980s (e.g., Lloyd, 1981; Blazewicz et al., 1986). Polynomially solvable cases are limited to a constant (fixed) range of task sizes, while the problem in its general form (arbitrary task sizes) is known to be NP-hard in the strong sense for an arbitrary number processors (Lloyd, 1981). The objective of mean flow time minimization is considered, for example, by Drozdowski and Dell'Olmo (2000) who also prove the NP-hardness of problem $P \mid size_i; p_i = 1 \mid \sum C_i$.

Given those unit time task slices of predefined size, the analogy to bin packing is obvious (Garey et al., 1976; Coffman et al., 1978). The uniform character of the chains is in fact reminiscent of what is known as *high-multiplicity* bin packing (Gabay, 2014). As long as the number of distinct sizes is a fixed constant, such type of problem can be solved in polynomial time (Goemans & Rothvoß, 2014; Jansen, 2017).

Rigid parallel tasks on multiprocessor resources can also be linked by chain precedence constraints. Blazewicz and Liu (1996) show that scheduling arbitrary chains of rigid unit-time tasks on three processors, that is, $P3 \mid size_i; p_i = 1; chains \mid C_{\max}$, is strongly NP-hard. Some special cases, like uniform chains involving only tasks with $size_i \in \{1, k\}$ ($k \leq m$ arbitrary), are solvable in polynomial time.

While maximum time lags do not seem to be an issue in classical, single-resource multiprocessor scheduling itself, they received at least some attention in related fields, like resource-constrained project scheduling (see, e.g., Schutt et al., 2013). Although (sparse) research has been conducted on bin packing with precedence constraints (Dell'Amico et al., 2012; Pereira, 2016), time lags, and in particular those of maximum-type, did not play a role so far. Scholl et al. (2010) include them as distance constraints in their model formulation of a generalized assembly line balancing problem, but they drop them later when it comes to problem solving due to a lack of practical relevance.

4 Adopting a packing-oriented perspective

As already indicated in Sect. 3, an analogy can be established between the scheduling problem at hand and one-dimensional bin packing. Each slice can be directly mapped to an item. Its weight $w_{ij} = size_{ij}$ and thus simply w_i (see Sect. 2). To keep the notation consistent throughout the paper, the term slice will be continued to be used, also in the packing context. Each bin k corresponds to a time period of the original scheduling problem, imposing an absolute order among the bins. In other words, it actually matters to *which* of the available bins a slice is allocated. The number of available bins is denoted by $\bar{k}$, representing the length of the planning horizon. A schedule with starting times s_{ij} for each slice can be mapped to a bin packing (and vice versa) in the following manner: a slice with $s_{ij} = 0$ is assumed to be placed in the first bin, a slice with $s_{ij} = 1$ in the second bin, and so on. Given the order among the bins, it is easy to also transfer the chain-like precedence constraints from the scheduling problem, including the minimum and maximum delays. All delays can simply be represented by "distances" between numbered bins. Although all the developed concepts would basically support varying bin capacities, one and the same capacity $\mathcal{C}$, equal to the number of processors m, is used for all bins.

This section first introduces some fundamental concepts and symbol notation that is used throughout the paper. Then a packing-based mixed integer programming (MIP) formulation is presented, modeling the precedence constraints in a non-standard fashion. The given model also constitutes the comparison baseline for the computational experiments in Sect. 8.

To allow for an easier look-up, the symbols that are used in the remainder of the paper are summarized in Tables 1 (constants) and 2 (sets & functions).

Table 1 Symbol notation: constants

m	Number of processors.
$size_{i,j}$	Size/width of slice j of chain i
n_i	Length of chain i (= number of slices in the chain)
$C_{i,j}$	Completion time of slice j of task i
$s_{i,j}$	Starting time of slice j of task i
$w_{i,j}$	Weight of task slice (i, j)
w_i	Base weight of chain i
$\bar{k}$	Length of the time horizon
$\mathcal{C}$	Bin capacity
n'_i	Rem. number of slices for chain i (in a partial packing)

Table 2 Symbol notation: sets & functions

T	Set of (outer) tasks
S	Feasible schedule, defined as a set of starting times
$\mathscr{F}_k$	Feasible set packed into bin k
$\mathbb{F}^{\bar{k}}$	Set of all potential feasible packings fitting into $\bar{k}$ bins
h_i	Multiplicity of chain i slices in a feasible set
$\ell(F_k)$	Function calculating the total load of a feasible set F_k
$\underline{F}_k$	Compulsory part of feasible set F_k
$\underline{F}^*$	Intersection of all compulsory parts of all packings
$\tilde{\underline{F}}$	Approximation of $\underline{F}^*$
$\tilde{T}_k$	Set of in-progress chains before packing bin k
$\underline{r}(i,k)$	Function yielding a lower bound on the number of remaining slices for chain i before packing bin k
$r(F,i,k)$	Function yielding the true number of remaining slices for a chain i and a bin k, based on a feasible packing F
$f(F)$	Function returning the usage cost of a packing F
$\mathscr{F}^s_k$	Set of slice-saturated feasible sets packable in bin k
$\mathscr{F}^e_k$	Set of extensible feasible sets packable in bin k
$\mathscr{F}^m_k$	Set of maximal feasible sets packable in bin k

4.1 Conceptual groundwork

The following definition of a *feasible set* is fundamental to all problem-specific concepts presented in this paper.

Definition 1 A set $F_k = \{(1, h_1), (2, h_2), \ldots, (n, h_n)\}$, with $h_i \in \mathbb{Z}^{\geq 0}$ reflecting the slice count (or multiplicity) of chains $i = 1, \ldots, n$ in the corresponding packing is called a *feasible set* for bin k, $1 \leq k \leq \bar{k}$, if the resulting total load does not exceed the capacity $\mathcal{C}$ of the bin. Removing a pair $(i', h_{i'})$ from a feasible set is equivalent to setting the multiplicity of chain i' to zero, hence, $F_k \setminus \{(i', h_{i'})\} = \{\ldots, (i', 0), \ldots\}$.

For reasons of convenience, F_k will also be used as a function henceforth. In fact, $F_k : T \to \mathbb{Z}^{\geq 0}$ without changing the semantics.

Let $\mathscr{F}_k$ denote the set of all feasible sets fitting into bin k. Then $\mathbb{F}^{\bar{k}} = \mathscr{F}_1 \times \mathscr{F}_2 \times \ldots \mathscr{F}_{\bar{k}}$ is the set of all potential, feasible *packings* that fit into $\bar{k}$ bins for a given instance.

The total load of a feasible set is an important quantity that frequently occurs in the upcoming presentation of the advanced solution-oriented concepts. To simplify the notation, let $\ell : \{A \mid A \subset \{(i, h_i) \mid i \in T, 1 \leq h_i \leq n_i\}\} \to \mathbb{Z}^{\geq 0}$ denote the function that calculates the total load of a feasible set A as $\ell(A) := \sum_{(i,h_i)\in A} h_i \cdot w_i$. The set $\{(i, h_i) \mid i \in T, 1 \leq h_i \leq n_i\}$ used in the definition of function ℓ represents the set of all feasible sets for a particular problem instance.

4.2 A packing-based mixed integer programming formulation

Apart from the precedence constraints, the mixed integer programming formulation can be stated in a straightforward manner. It is based on binary decision variables $x_{ijk} \in \{0, 1\}$ that indicate whether a slice (i, j) is assigned to bin k or not:

$$\min \quad \sum_{k=1}^{\bar{k}} k \cdot \sum_{i\in T} \sum_{j=1}^{n_i} x_{ijk} \cdot w_i \tag{3}$$

$$\text{s.t.} \quad \sum_{i\in T} \sum_{j=1}^{n_i} x_{ijk} \cdot w_i \leq \mathcal{C} \quad \forall\, 1 \leq k \leq \bar{k}, \tag{4}$$

$$x_{i,j-1,1} \geq x_{i,j,1} \quad \begin{array}{l} \forall\, i \in T, \\ \forall\, 2 \leq j \leq n_i, \end{array} \tag{5}$$

$$x_{i,j-1,k-1} + x_{i,j-1,k} \geq x_{i,j,k} \quad \begin{array}{l} \forall\, i \in T, \\ \forall\, 2 \leq j \leq n_i, \\ \forall\, 2 \leq k \leq \bar{k}, \end{array} \tag{6}$$

$$\sum_{k=1}^{\bar{k}} x_{ijk} = 1 \quad \begin{array}{l} \forall\, i \in T, \\ \forall\, 1 \leq j \leq n_i, \end{array} \tag{7}$$

$$x_{ijk} \in \{0, 1\} \quad \begin{array}{l} \forall\, i \in T, \\ \forall\, 1 \leq j \leq n_i, \\ \forall\, 1 \leq k \leq \bar{k}. \end{array} \tag{8}$$

The objective function (Eq. (3)) is essentially the *linear usage cost* objective known from Braune (2019). Constraints (4) ensure that the bins' capacities are not exceeded. Constraints (5) and (6) enforce the minimum and maximum time lags by exploiting the semantics of the binary variables. The idea is that if a slice (i, j) is assigned to a bin k, then its predecessor in the chain, i.e., $(i, j-1)$, must be assigned either to the previous bin or the same bin as (i, j). This way of modeling the precedence constraints turned out to be superior to the conventional approach, derived from Eqs. (1) and (2), as indicated by preliminary computational experiments. Constraints (7) finally make sure that each slice is assigned to exactly one bin.

5 Tightening lower bounds

Various custom lower bounds for a bin packing problem with linear usage cost have recently been proposed in Braune (2019). The linkage to this kind of problem is pointed out in Sect. 4. The main difference, on the other hand, is the absence of precedence constraints in the referenced problem.

In this section, an approach is proposed to exploit those constraints with regard to tightening the existing lower bounds. The resulting lower bounds are "local" in the sense that they have to be recomputed each time a partial packing (or schedule) has been fixed. In fact, some slices have to be already packed for precedence constraints to take effect. The maximum time lags then lead to a situation where a certain part of the packing in yet unpacked bins is already predetermined. Suppose that a particular partial packing has been fixed and that the bins are re-indexed such that the last packed bin and the first unpacked bin receive the indices 0 and 1, respectively. Assume further that at this state of packing, a maximum number of *additional* $\bar{k}$ bins can still be packed in a way such that no trivial improvements can be made without violating the precedence constraints.

The *compulsory part* of a feasible set F_k (see Definition 1 in Sect. 4.1) is given by $\underline{F_k} = \{(i, 1) \mid i \in T \wedge F_k(i) \geq 1 \wedge F_{k-1}(i) \geq 1\}$ and hence consists of exactly one slice for each chain which is also packed into the previous bin and is not yet finished, meaning that it has slices left to be packed. The notion of a compulsory part directly follows from the precedence constraints with maximum time lags. It means that the underlying feasible set could be altered by shifting slices to earlier or later bins, but its compulsory part must remain untouched in order not to violate the non-preemption constraints.

Since a packing $F \in \mathbb{F}^{\bar{k}}$ can be considered as a vector of feasible sets, one for each bin $1 \leq k \leq \bar{k}$, the compulsory part $\underline{F}$ of a packing can be defined in an analogous way.

If the intersection of all compulsory parts of all potential feasible packings, denoted by $\underline{F}^*$, was known, a notable portion of the unpacked bins could already receive some fixed load up front, regardless of which packing is implemented afterward. Figure 3 illustrates the idea, based on a concrete example. The area below the bold, dotted line represents the common compulsory part $\underline{F}^*$, pretending that it is known. The shaded in dark grey represents an approximation of $\underline{F}^*$, as it will be proposed below. Finally, the feasible set composition around that part is just one particular option to complete the packing.

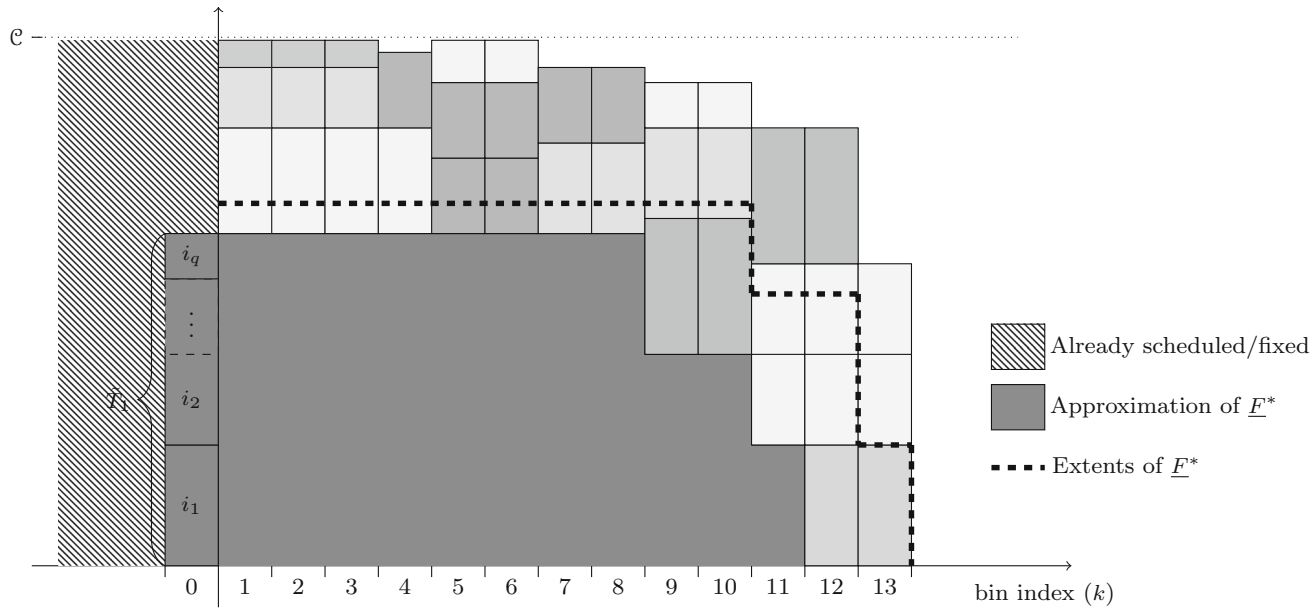

Fig. 3 The concept of a common compulsory part and its approximation, based on maximum time lags

In terms of the already mentioned lower bounds, such a fixed, rigid block might ideally lead to less fractional load and thus to tighter bounds in general. Unfortunately, the determination of $\underline{F}^*$ would require the enumeration of all feasible sets and is thus computationally prohibitive. Therefore, the remainder of this section is dedicated to an approximation of $\underline{F}^*$.

The main challenges of developing such an approximation, referred to as $\tilde{\underline{F}}$ henceforth, are (9) to ensure that the lower bound property is not violated and (10) to keep the fixed area as large as possible to ensure a proper tightening effect. The key to the computation of $\tilde{\underline{F}}$ is to focus on chains that are *in progress* at the transition between packed and unpacked bins. A chain is considered in progress if it has at least one unpacked slice remaining. In fact, the remaining number of slices of in-progress chains determines the shape of the rigid load that extends into the unpacked bins. The proposed approximation is therefore based on determining the *minimum* possible number of remaining slices for each unpacked bin $1 \leq k \leq \bar{k}$.

From a more general point of view, let $\tilde{T}_k$ denote the set of in-progress chains *before* packing bin k. Assuming that this property holds for q chains after packing the most recent bin, $\tilde{T}_1 = \{i_1, i_2, \ldots, i_q\}$ is predetermined and thus constant, as illustrated in Fig. 3. Accordingly, the first fragment of the approximation, $\tilde{\underline{F}}_1 = \{(i, 1) \mid i \in \tilde{T}_1\}$, can also be fixed. All subsequent sets $\tilde{T}_k$, with $k \geq 2$, again depend on the actual packing and again cannot be determined without excessive computational effort. However, it is possible to approximate the sets $\tilde{T}_k$: let $\underline{r}(i, k)$ denote a function yielding a lower bound on the number of *remaining* slices for chain i *before* packing bin k. Based on the (known) number of remaining slices n'_i at the boundary between the packed and the unpacked areas, a recursive definition of function $\underline{r}$ can be given as follows:

$$\underline{r}(i,k) = \begin{cases} n'_i, & \text{for } k = 1, \\ \underline{r}(i,k-1) - \left\lfloor \dfrac{\mathcal{C} - \sum\limits_{i' \in \tilde{T}_1, i' \neq i} \min(1, \underline{r}(i',k-1)) \cdot w_{i'}}{w_i} \right\rfloor, & \text{for } k \geq 2. \end{cases} \tag{9}$$

The idea of the approach is to use up all the residual space of a bin for slices of a particular chain $i \in \tilde{T}_1$. This is done for each chain separately and independently from the other chains. Hence, the residual space is essentially used q times, which constitutes a relaxation of the capacity constraints. An approximation set $\tilde{\underline{F}}_k$ can then be defined as

$$\tilde{\underline{F}}_k = \begin{cases} \{(i,1) \mid i \in \tilde{T}_1\}, & \text{for } k = 1, \\ \{(i, \min(1, \underline{r}(i,k))) \mid i \in \tilde{T}_1\}, & \text{for } 2 \leq k \leq \bar{k}. \end{cases} \tag{10}$$

Lemma 1 *Let $r : \mathbb{F}^{\bar{k}} \times T \times \{1, \ldots, \bar{k}\} \to \mathbb{Z}^{\geq 0}$ denote a function that yields the true number of remaining slices for a chain $i \in T$ and bin k, $1 \leq k \leq \bar{k}$, based on a feasible packing $F \in \mathbb{F}^{\bar{k}}$. Again, $r(F, i, 1) = n'_i$, and for $k \geq 2$,*

$$r(F,i,k) = n'_i - \sum_{\kappa=1}^{k-1} F_\kappa(i). \tag{11}$$

Then

$$\forall\, F \in \mathbb{F}^{\bar{k}}, \forall\, i \in \tilde{T}_1, \forall\, 1 \le k \le \bar{k}: \quad \underline{r}(i,k) \le r(F,i,k). \tag{12}$$

Proof By induction on k.

1. Base case: $k = 1$ Since the packing is fixed and known before the first unpacked bin, so is n'_i, for all $i \in T$. Hence, for this bin, the remaining number of slices is independent of any packing $F \in \mathbb{F}^{\bar{k}}$, and thus clearly $\underline{r}(i,1) = r(F,i,1) = n'_i$, for all $i \in T$.
2. Induction hypothesis: Assume that for all $F \in \mathbb{F}^{\bar{k}}$ and $i \in \tilde{T}_1$, and for some $1 \le k \le \bar{k}$, $\underline{r}(i,k) \le r(F,i,k)$ holds.
3. Induction step: It is to be shown that $\underline{r}(i,k+1) \le r(F,i,k+1)$. To simplify the proof, a recursive definition of function r is given as follows ($k \ge 2$, because $r(F,i,1) = n'_i$):

$$r(F,i,k) = r(F,i,k-1) - F_{k-1}(i). \tag{13}$$

Making use of function ℓ from Sect. 4.1, returning the total load of a feasible set, it follows directly from Eq. (13) that

$$\begin{aligned}
r(F,i,k+1) &= r(F,i,k) - F_k(i) \\
&= r(F,i,k) - \frac{\ell(F_k) - (\ell(F_k) - F_k(i)\cdot w_i)}{w_i} \\
&\ge r(F,i,k) - \left\lfloor \frac{\mathcal{C} - (\ell(F_k) - F_k(i)\cdot w_i)}{w_i} \right\rfloor \\
&\ge r(F,i,k) - \left\lfloor \frac{\mathcal{C} - \sum_{i' \in \tilde{T}_1, i' \ne i} \min(F_k(i'), r(F,i',k))\cdot w_{i'}}{w_i} \right\rfloor \\
&\overset{\text{(by assumption)}}{\ge} \underline{r}(i,k) - \left\lfloor \frac{\mathcal{C} - \sum_{i' \in \tilde{T}_1, i' \ne i} \min(1, \underline{r}(i',k))\cdot w_{i'}}{w_i} \right\rfloor \\
&= \underline{r}(i,k+1).
\end{aligned}$$

□

Lemma 1 allows to finally prove the lower bound property of the proposed approximation $\underline{\tilde{F}}$.

Theorem 1 *Given a set of in-progress chains $\tilde{T}_1 \ne \emptyset$ resulting from a partial packing and an arbitrary completion $F \in \mathbb{F}^{\bar{k}}$ ($\bar{k} \ge 2$) of the (overall) packing, then for an approximation $\underline{\tilde{F}}$ as defined in Eq. (10),*

$$\ell(F_k) \ge \ell(\underline{\tilde{F}}_k), \tag{14}$$

for all $1 \le k \le \bar{k}$.

Proof For $k = 1$, the inequality trivially holds, because F_1 must contain all in-progress chains as well, with multiplicities greater than or equal to 1. Consider the case $2 \le k \le \bar{k}$:

$$\begin{aligned}
\ell(F_k) &= \sum_{i \in T} F_k(i)\cdot w_i \\
&\ge \sum_{i \in T} min(1, F_k(i))\cdot w_i \\
&= \sum_{i \in T} min(1, F_k(i), r(F,i,k))\cdot w_i \\
&\ge \sum_{i \in \tilde{T}_1} min(1, F_k(i), r(F,i,k))\cdot w_i.
\end{aligned}$$

Based on the fact that $F_k(i) \ge 1$ as long as $r(F,i,k) \ge 1$, for all $i \in \tilde{T}_1$, and Lemma 1, one finally obtains

$$\begin{aligned}
&\sum_{i \in \tilde{T}_1} min(1, F_k(i), r(F,i,k))\cdot w_i \\
&= \sum_{i \in \tilde{T}_1} min(1, r(F,i,k))\cdot w_i \\
&\overset{(12)}{\ge} \sum_{i \in \tilde{T}_1} min(1, \underline{r}(i,k))\cdot w_i \\
&= \ell(\underline{\tilde{F}}_k).
\end{aligned}$$

□

6 Dominance and symmetry breaking rules

6.1 Dominance between alternative packings

This section deals with equivalence and dominance relations between alternative packings of the same set of task slices. First, it is shown that the precedence constraints do not interfere with these relations. This is a fortunate side effect of the special kind of maximum time lags present in the problem. Second, a set of conditions is presented by which it is possible to decide whether a certain packing can be dropped from consideration because it cannot lead to a better solution than an alternative one.

Lemma 2 *Consider two packings $F \in \mathbb{F}^{\bar{k}}$ and $F' \in \mathbb{F}^{\bar{k}'}$, involving the same set of slices (and thus chains) and allocating $\bar{k}$ and $\bar{k}'$ bins (starting from bin 1), respectively. More formally,*

$$\forall\, i \in T\colon \sum_{k=1}^{\bar{k}} F_k(i) = \sum_{k=1}^{\bar{k}'} F'_k(i). \tag{15}$$

Then F and F' are equivalent *with respect to the precedence constraints, meaning that exactly the same restrictions regarding chain continuation apply to both packings.*

Proof Packings F and F' can be partial packings, that is, some chains may still have slices remaining to be packed starting from bins $\bar{k}+1$ and $\bar{k}'+1$, respectively. It follows from Eq. (15) that the number of *remaining* slices n'_i for each chain $i \in T$ has to be the same in both cases. Two major cases are distinguished:

1. Packing F is self-contained in the sense that all started chains are also finished within F. This must hold for F' as well, because otherwise the set of slices would be different.
2. Packing F contains incomplete chains.
   Then exactly the same chains are also incomplete in F', with the same number slices left for each of those chains.

□

Let $f : \mathbb{F}^{\bar{k}} \to \mathbb{R}$ denote the objective function, returning the usage cost of a particular packing $F \in \mathbb{F}^{\bar{k}}$, that is, $f(F) := \sum_{k=1}^{\bar{k}} k \cdot \ell(F_k)$. It is easy to see that the definition of f is in accordance with the objective function of the MIP formulation provided in Sect. 4.2, with variables x_{ijk} essentially deciding whether a slice (i, j) is part of a feasible set F_k or not. To prove the dominance between alternative packings of the same slices, the following result is needed:

Lemma 3 *Consider a particular packing $F \in \mathbb{F}^{\bar{k}}$, involving $\bar{k}$ bins. Delaying the packing by κ bins increases the objective function value $f(F)$ by κ times the total load of slices contained in F.*

Proof Without loss of generality, it is assumed that F starts in bin 1. Let F' be the packing resulting from the shift, simply containing a load of zero in the first κ bins and then the original packing F starting in bin $\kappa+1$. Consequently, $\bar{k}' = \bar{k}+\kappa$. The difference between $f(F')$ and $f(F)$ is then given by

$$\begin{aligned} f(F') - f(F) &= \sum_{k=\kappa+1}^{\bar{k}'} k \cdot \ell(F_{k-\kappa}) - \sum_{k=1}^{\bar{k}} k \cdot \ell(F_k) \\ &= (\kappa+1)\cdot\ell(F_1) + \cdots + (\kappa+\bar{k})\cdot\ell(F_{\bar{k}}) \\ &\quad - 1\cdot\ell(F_1) - \cdots - \bar{k}\cdot\ell(F_{\bar{k}}) \\ &= \kappa\cdot\ell(F_1) + \cdots + \kappa\cdot\ell(F_{\bar{k}}) = \kappa\cdot\sum_{k=1}^{\bar{k}}\ell(F_k). \end{aligned}$$

□

Theorem 2 *Let F and F' be two alternative packings involving the same set of task slices, as specified in Lemma 2. Packing F cannot lead to a better solution than F' if one of the following conditions holds:*

$$f(F) \geq f(F') \wedge \bar{k} \geq \bar{k}' \tag{16}$$

$$f(F) < f(F') \wedge \bar{k} \geq \bar{k}' \wedge f(F') - f(F) \leq (\bar{k}-\bar{k}')\cdot\sum_{i\in T} n'_i \cdot w_i \tag{17}$$

$$f(F) \geq f(F') \wedge \bar{k} < \bar{k}' \wedge f(F) - f(F') \geq (\bar{k}'-\bar{k})\cdot\sum_{i\in T} n'_i \cdot w_i \tag{18}$$

Proof Lemma 2 guarantees that the precedence constraints do not interfere with any of the above conditions. It is then easy to see that F cannot lead to a better solution than F' if both, the objective value and the number of bins are at least as high as for F'. The remaining total load is exactly the same for both packings and it will be allocated to the still unoccupied bins after the respective partial load. The optimal packing for this remaining load will therefore also be exactly the same in both cases.

Conditions (17) and (18) are less intuitive, however. Figure 4 illustrates the rationale behind them. It shows two different packings for the same set of chains (and slices). The packing on the left occupies eight bins and yields a usage cost (TWCT) objective value of 900. The packing on the right uses one bin more, but leads to a slightly lower objective value of 897. This peculiarity of the problem is in fact the key to the two additional dominance rules.

Condition (17) refers to the case where F looks similar to the right part of Fig. 4 and thus occupies more bins than F' at lower usage costs. It then solely depends on the amount of the remaining (unpacked) load, whether F is nevertheless dominated by (or at least as good as) F' or not. Let $\bar{k}''$ denote the number of *additional* bins that can be allocated by those remaining slices. Any packing $F'' \in \mathbb{F}^{\bar{k}''}$ could be started $(\bar{k}-\bar{k}')$ bins earlier when adopting F'. From Lemma 3, it is known that the objective value difference amounts to $(\bar{k}-\bar{k}')\cdot\sum_{k=1}^{\bar{k}''}\ell(F''_k) = (\bar{k}-\bar{k}')\cdot\sum_{i\in T} n'_i\cdot w_i$, based on the fact that F'' contains all remaining slices with multiplicities n'_i.

Condition (18) covers the exact opposite case. Here, F already induces a higher usage cost objective, but requires fewer bins. Hence, F might still lead to a better solution. This is precluded, however, if the difference in the objective value is larger than the savings that can be gained from the more compact way of packing. Any packing $F'' \in \mathbb{F}^{\bar{k}''}$ could be started $(\bar{k}'-\bar{k})$ bins earlier when adopting F instead of F'. The gain is therefore $(\bar{k}'-\bar{k})\cdot\sum_{k=1}^{\bar{k}''}\ell(F''_k) = (\bar{k}'-\bar{k})\cdot\sum_{i\in T} n'_i\cdot w_i$, according to Lemma 3. □

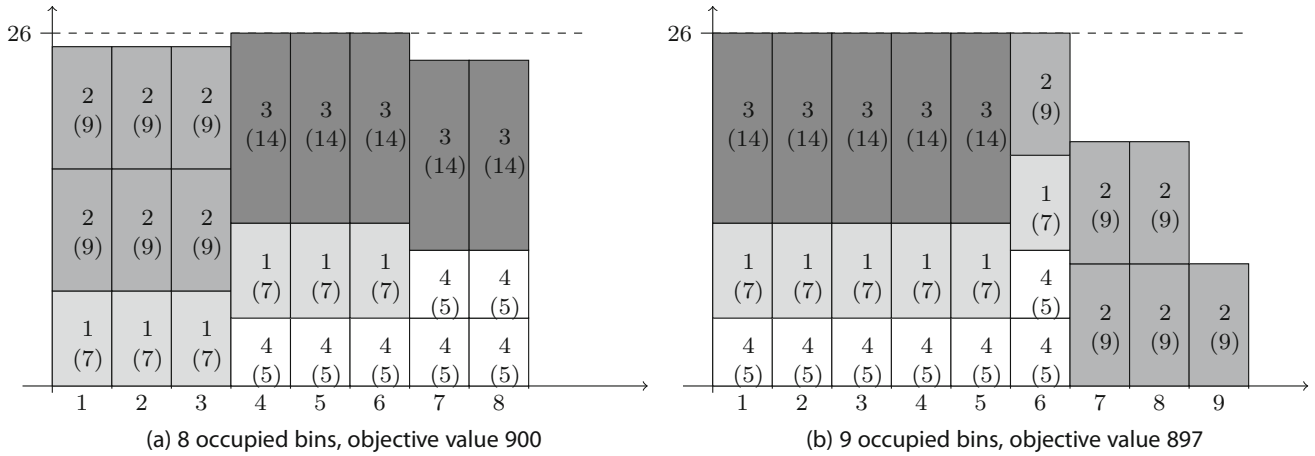

Fig. 4 Example problem with an optimal solution that occupies more bins than would be minimally necessary

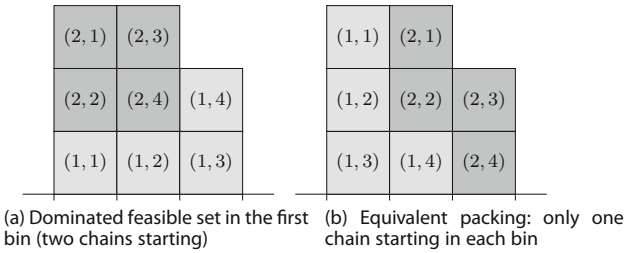

Fig. 5 Dominance between packings involving different chains with the same base weight

6.2 Equally weighted chains

Consider a problem instance involving two or more chains sharing the same base weight. During packing, a lot of symmetry might arise, as indicated in Fig. 5. In the given example, the feasible set in the first bin contains one slice of chain 1 and two slices of chain 2. An equivalent packing could be obtained by putting together two slices of chain 1 and one slice of chain 2. It is easy to see that the number of different ways to combine chains such kind in a feasible set increases with the number of chains and the number of times the (common) chain size fits into a bin. However, the following symmetry breaking rules help reduce that combinatorics considerably, without omitting potential optimal solutions. In a feasible set, for each distinct value of w_i,

- Only one chain is allowed to start,
- Only one chain is allowed to finish,
- Arbitrarily many chains are allowed to start *and* finish.

The last rule applies to chains that are short enough to entirely fit into a single bin. The right part of Fig. 5 exemplarily shows how the application of these rules leads to an equivalent packing. The equivalence directly follows from Theorem 2. The time of application of the associated rules is different though. While dominance (or equivalence) between packings involving the same set of slices can only be checked in retrospect, the above stated rules can be used to filter feasible sets up front, before they are even packed.

6.3 Extensible feasible sets

Definition 2 A feasible set F_k is called *extensible* if it leaves space for an additional slice of a chain that is *not* part of F_k when packed to a bin k. More formally, there exists a chain $i' \in T$, with $F_k(i') = 0$, such that $\sum_{i \in T} F_k(i) \cdot w_i + w_{i'} \leq \mathcal{C}$.

Unlike one would expect, extensible feasible sets are of vital importance when constructing optimal solutions for the problem at hand. Consider the example shown in Fig. 6. The packing on the left-hand side is an optimal one, featuring an extensible feasible set in the very first bin. There would be still enough space to pack a slice of chain 5 ($w_5 = 2$). However, due to the maximum time lags, chain 5 must be continued and would thus prevent the feasible set $\{(2, 2), (3, 1)\}$ from being packed in the subsequent bins. Hence, creating only "maximal" feasible sets, meaning that no further slice of any chain can be added to the feasible set without violating the capacity constraints, may not be sufficient for solving a given problem instance to optimality. Note that starting with the feasible set $\{(2, 2), (3, 1)\}$ makes things even worse, requiring one more bin and leading to an inferior solution as well. This is due to the fact that the feasible set $\{(3, 2)\}$ is no more packable in this situation (chain 2 is forced to be continued), and besides that, chain 3 is incompatible with chain 4.

Note that an extensible feasible set might actually occur also in the middle of an optimal packing, as shown in Fig. 7. In the depicted example, $F_9 = \{(5, 2), (1, 1)\}$ would leave enough space for starting chain 0 (size 2). However, proceeding like this prevents chain 2 from being packed together with chain 5, which is obviously necessary for constructing an optimal solution.

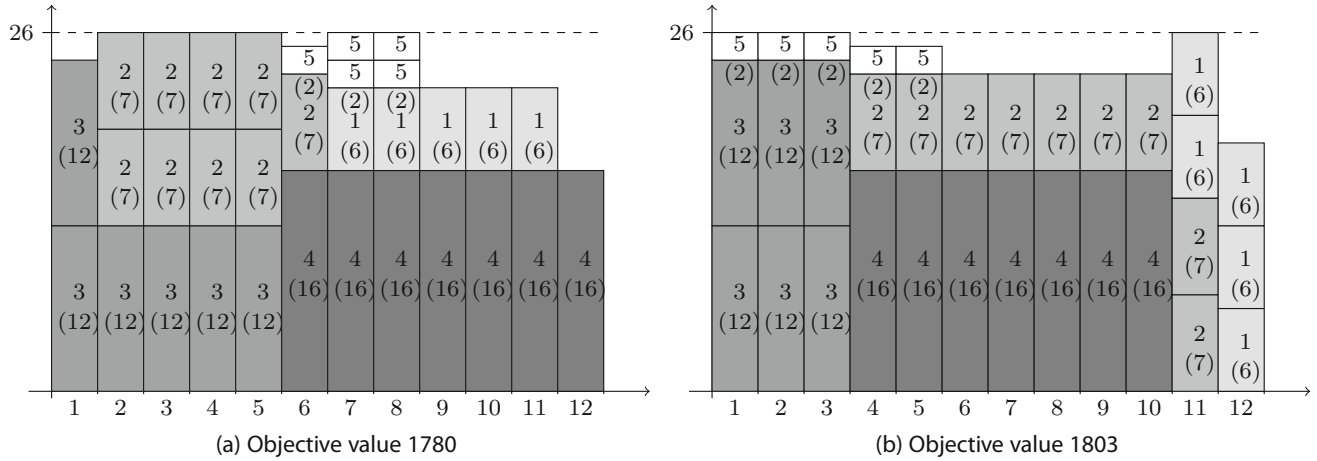

Fig. 6 Extensible versus maximal feasible sets at the beginning of a packing

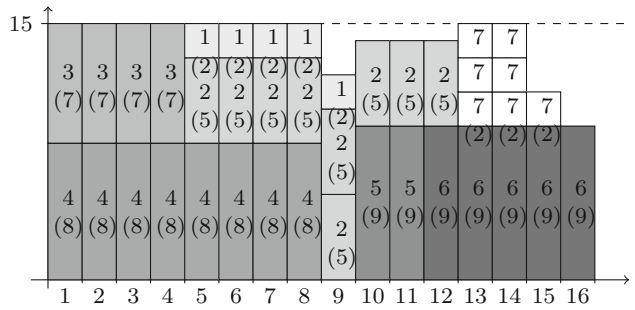

Fig. 7 An extensible feasible set amid an optimal packing

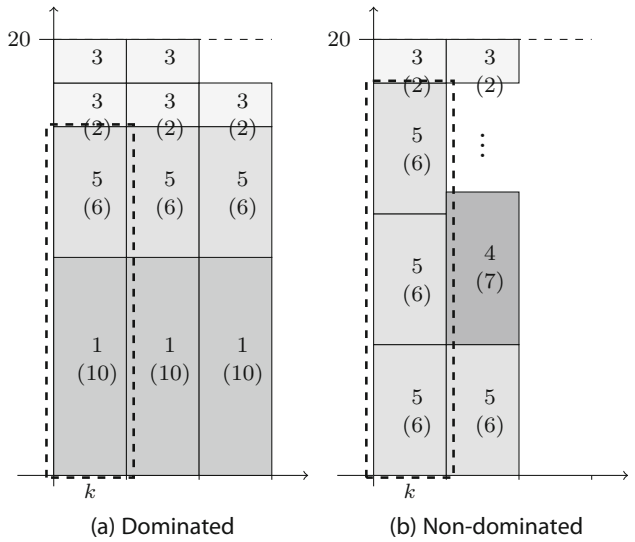

Fig. 8 Extensible feasible sets with varying dominance status

Under particular circumstances, however, it is possible to restrict the search space by discarding dominated extensible feasible sets or their extensions.

Proposition 1 *Consider an extensible feasible set F_k allocated to a bin k and assume that it is possible to allocate the same feasible set also to the next bin $k+1$. Then any feasible set F'_{k+1} for bin $k+1$ that is a* simple extension *of F_k, hence adding at least one slice of another chain according to Definition 2, cannot be part of an optimal solution.*

Proof Clearly, a packing containing F_k and its simple extension F'_{k+1} in directly consecutive bins can be trivially improved with regard to the usage cost objective by simply swapping the allocations of the two feasible sets, i.e., putting F'_{k+1} into bin k and F_k into bin $k+1$. □

Proposition 2 *Let F_k be an extensible feasible set assigned to bin k. The set F_k cannot be part of an optimal solution if both*

1. *F_k contains exactly one slice for each covered chain and*
2. *There exists an extending chain $i' \in T$ that can be finished before the continuation of F_k across the subsequent bins ends.*

Proof The first condition ensures that the *configuration* of F_k does not change over time. This means that F_k can only be extended and not reduced. The purpose of packing an extensible feasible set is to leave enough flexibility for switching to another feasible set (that is *not* a simple extension) in one of the subsequent bins, as depicted in Fig. 6. If F_k contains two or more slices for a chain, it is possible to switch to another feasible set in *any* bin, as long as at least one slice of each chain in progress is packed. A chain that extends F_k would then again impose restrictions on the transition to another feasible set, if it requires more than one bin to finish. This case is illustrated in the right part of Fig. 8. If, on the other hand, F_k contains exactly one slice for each covered chain, that is, when $F_k(i) = 1$ for all $i \in T$ with $F_k(i) \geq 0$, no such transition is possible unless at least one involved chain is running out of slices. If a fitting extension chain $i' \in T$

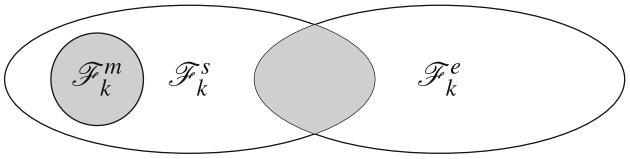

Fig. 9 Relations among different types of feasible sets

can be found that can be finished before this happens, F_k is clearly dominated by the resulting extension F'_k (see left part of Fig. 8). □

7 A branch-and-bound algorithm

The proposed branch-and-bound (B&B) algorithm constructs a schedule in a chronological fashion, starting at time period 0 (bin 1). Its core concept is to enumerate feasible sets for each time period or bin. From a packing-oriented point of view, it can therefore be classified as a "bin-completion" algorithm (Fukunaga & Korf, 2007).

Definition 3 A feasible set F_k is called *slice-saturated* if for none of its already included chains, an additional slice can be added without exceeding the capacity of bin k. Formally speaking, this is the case when there is no $i' \in T$ with $F_k(i') > 0$ and $\sum_{i \in T} F_k(i) \cdot w_i + w_{i'} \leq \mathcal{C}$.

If a partial packing has already been established, this property also depends on the remaining number of slices n'_i of each chain i. Of course, a feasible set that runs out of slices for one of its chains is also slice-saturated.

Definition 4 A feasible set F_k is called *maximal* if no additional slice of *any*, not necessarily already included, chain can be added without exceeding the capacity of bin k.

Note that a slice-saturated feasible set can still be extensible (see Definition 2), because there may be enough space for a slice of a chain that is not yet included. On the other hand, an extensible feasible set does not necessarily have to be slice-saturated. Let $\mathscr{F}_k^s$, $\mathscr{F}_k^e$ and $\mathscr{F}_k^m$ denote the set of slice-saturated, extensible and maximal feasible sets packable in bin k, respectively. Then the following set relations hold: $\mathscr{F}_k^m \subset \mathscr{F}_k^s$ and $\mathscr{F}_k^m \cap \mathscr{F}_k^e = \emptyset$.

As already indicated in Sect. 6.3, in order to find an optimal solution to the problem at hand, it is not sufficient to restrict the search to *maximal* feasible sets. Rather, the search has to be extended to sets that are extensible *and* slice-saturated. An optimization algorithm therefore has to consider the set union $\mathscr{F}_k^o = \mathscr{F}_k^m \cup (\mathscr{F}_k^s \cap \mathscr{F}_k^e)$, as depicted in Fig. 9 by the areas shaded in light gray. Note that a feasible set that is not extensible but slice-saturated has to be a maximal one.

Proposition 3 *Consider an optimal solution to a particular instance of the discrete malleable task scheduling problem with TWCT objective, allocating k^* bins. Then in all bins $k \in \mathbb{N}^+$, $k \leq k^*$, this solution contains either a maximal or an extensible, slice-saturated feasible set.*

Proof Suppose that in one of the bins $1 \leq k \leq k^*$, the solution contains a feasible set that is not maximal and not slice-saturated. Then it is easy to see that the solution can be trivially improved by removing a slice from a later (e.g., the next) bin and adding it to this feasible set. No such move operation would violate the precedence constraints because at least the immediately succeeding bin contains another slice of one of the chains in question. Note that any feasible set occurring in the last bin k^* is always a maximal one (all remaining slices are simply put there), otherwise the solution would not be optimal. □

Every node in the B&B tree corresponds to a partial packing up to a particular bin $\hat{k}$ (starting from bin 1). Branching is conducted by determining the set of potential feasible sets for bin $\hat{k}+1$, that is, $\mathscr{F}^o_{\hat{k}+1}$. Hence, each branch is an extension of the current partial packing by exactly one additional (packed) bin. To avoid the exploration of non-promising regions of the search space, a variety of techniques is used to filter the potential feasible sets. The most important concepts have already been presented in Sects. 5 and 6 and will therefore be discussed with regard to their embedding into the B&B algorithm only. It must be emphasized at this point that conventional techniques for filtering feasible sets in bin packing, like the ones proposed by Martello and Toth (1990) and Scholl et al. (1997), cannot be applied to the problem at hand. The interchange arguments used in the corresponding proofs are not generally valid under the presence of precedence constraints.

Algorithm 1 gives an outline of the flow structure of the proposed B&B algorithm. To keep it short, it is an informal, high-level description, leaving out implementation details. Rather, its main goal is to illustrate the sequence in which the major steps of the algorithm (bounding, filtering and consistency checking) are executed. The three filtering procedures called at the beginning (lines 2, 3, and 4), are not further outlined. The corresponding techniques have been introduced in Sects. 6.2 and 6.3. Note that the incumbent solution and its associated objective value UB are treated as global variables here (lines 14 and 20). It is assumed that they are initialized before the algorithm is run, by using a constructive heuristic, for example.

7.1 Bounding

Assume that a particular feasible set $F_{\hat{k}+1} \in \mathscr{F}^o_{\hat{k}+1}$ has been chosen to extend the existing partial packing. A lower bound on the usage cost objective obtained from the resulting new partial packing can be used to decide whether the extension

Algorithm 1: Outline of the proposed branch-and-bound procedure.

Input : A partial packing F, and the index of its latest packed bin $\hat{k}$.

Determine potential feasible sets for bin $\hat{k}+1$: $\mathscr{F}^{o}_{\hat{k}+1}$;
$\mathscr{F}^{o'}_{\hat{k}+1} \leftarrow$ `FilterEqualWeightChains(`$\mathscr{F}^{o}_{\hat{k}+1}$`);` `// Sec. 6.2`
$\mathscr{F}^{o'}_{\hat{k}+1} \leftarrow$ `FilterFeasSetExtensions(`$\mathscr{F}^{o'}_{\hat{k}+1}$`);` `// Prop. 1`
$\mathscr{F}^{o'}_{\hat{k}+1} \leftarrow$ `FilterExtFeasibleSets(`$\mathscr{F}^{o'}_{\hat{k}+1}$`);` `// Prop. 2`
foreach $F_{\hat{k}+1} \in \mathscr{F}^{o'}_{\hat{k}+1}$ **do**
  Create packing F' by appending $F_{\hat{k}+1}$ to F;
  if *F' is not complete* **then**
    Perform constraint propagation on F' (by CP model);
    if *CP model is consistent and no alternative dominating packing F'' exists in no-goods* **then**
      Set up problem instance I for remaining slices;
      Create tightened instance I' of I using compulsory part;
      Compute lower bound LB based on I';
      if $LB < UB$ **then**
        `BranchAndBound(`$F', \hat{k}+1$`);`
        Add F' to no-goods memory structure;
      end
    end
  else `/*Leaf node reached(new incumbent?) */`
    if $f(F') < UB$ **then**
      $UB \leftarrow f(F')$;
      Update incumbent solution (data);
    end
  end
  Remove $F_{\hat{k}+1}$ from F';
end

$F_{\hat{k}+1}$ can safely be dropped from consideration or not. For this purpose, the bounding scheme $L^{LUC}_{2\text{LLM}}$ that has proven particularly successful for the linear usage cost bin packing problem without precedence constraints (Braune, 2019) is employed in conjunction with the tightening approach presented in Sect. 5.

The bound computation is based on a partial problem that consists of all slices (and chains) left to be packed after the extension $F_{\hat{k}+1}$ has been added. Hence, $F_{\hat{k}+1}$ is the last bin of the packed area shown in Fig. 3. To incorporate the approximation $\underline{\tilde{F}}$ of the compulsory part of the continuation of the packing into the bound computation, the following approach is used: The corresponding partial problem is made up of two different kinds of slices. First, a dummy slice is created for each bin that is occupied by the approximation $\underline{\tilde{F}}$. The weight of each such dummy slice equals the load of $\underline{\tilde{F}}$ in the corresponding bin. Second, slices of chains that are in-progress or still completely unpacked are added to the problem. If a compulsory part exists, that is, when at least $\underline{\tilde{F}}_1 \neq \emptyset$, some of the latter slices are already tied to $\underline{\tilde{F}}$. Note that by the definition of $\underline{\tilde{F}}$ [see Eq. (10)], it contains at most one slice of each in-progress chain per bin. Let n''_i denote the number of remaining slices of a chain $i \in T$ that are *not* part of $\underline{\tilde{F}}$. One obtains $n''_i = n'_i - (\min\{k \in \mathbb{N} \mid \underline{r}(i,k) = 0\} - 1)$ for $i \in \tilde{T}_1$, and simply $n''_i = n'_i$ otherwise. Note that in this specific context, k is relative to the beginning of the unpacked area, hence, $k = 1$ for the first bin after the last packed bin $\hat{k}+1$.

Once the partial problem is set up based on the compulsory part and the remaining slices, it is passed to the bounding procedure. Both, dummy and regular slices can then be handled uniformly according to their weight and assigned to different sets, as typical for advanced bin packing lower bounds based on item grouping. The only major modification required is to always process the dummy slices (or the bins that contain them) first, with regard to matching and the distribution of fractional loads. In other words, to make effective use of the notion of compulsory parts, it has to be ensured that the corresponding dummy slices always appear at the beginning of any packing that is constructed by a lower bounding procedure.

7.2 Filtering

– *No-good recording* Section 6.1 covers the equivalence (and dominance) between alternative packings of the same set of slices. To exploit this concept, the proposed B&B algorithm stores packings that have not been dominated so far in a memory structure (a hash table). The contents of this memory structure are referred to as *no-goods*. This term has its roots in constraint programming (see, e.g., Schiex and Verfaillie, 1994)). In its original form, it refers to variable assignments that do not lead to a feasible solution of the associated constraint satisfaction problem and thus trigger a backtrack. In the optimization context, a no-good can be seen as a collection of solution characteristics that do not (immediately) lead to an optimal solution. The contributions by Jouglet (2002) and Jouglet et al. (2004) were among the first ones to introduce this concept in single-machine branch-and-bound methods, while Morrison et al. (2016) give a more general classification in the context of a survey on branch-and-bound techniques. For the problem at hand, every time a new partial packing is constructed, it is checked whether another packing involving the same slices has already been recorded. If one of the conditions of Proposition 2 [see Eqs. (16), (17) and (18)] succeeds to verify, then the new partial packing and thus its corresponding node can be discarded.

– *Symmetry breaking and dominance* The feasible set $F_{\hat{k}+1}$ is examined for symmetry if it contains slices of equal weight (see Sect. 6.2). If $F_{\hat{k}+1}$ is a simple extension of $F_{\hat{k}}$, it can immediately be discarded (see Sect. 6.3, Proposition 1). On the other hand, if $F_{\hat{k}+1}$ is an extensible feasible set itself, the conditions of Proposition 2 have to be checked to decide whether the node can be fathomed or not.

7.3 Supporting techniques

This section covers two auxiliary/supporting techniques providing information that can be exploited for intensified pruning of the B&B tree. First, the B&B algorithm carries a constraint programming (CP) model that is synchronized with the branching progress and can be used to detect inconsistent states. Second, through the (required) knowledge of all potentially extending feasible sets, the bin capacities might be "virtually" reduced, ideally leading to tighter lower bounds and an increased degree of constraint propagation.

– *Constraint propagation* Cambazard et al. (2013) proposed a constraint propagation approach in the context of linear usage cost bin packing. The incorporation of precedence constraints into that approach turned out to be impossible to do in an immediate way and is considered beyond the scope of this paper. Nevertheless, the propagation routine for the relaxed version of the problem (without precedence constraints) proved successful in reducing the B&B search effort. What the routine essentially does is to iteratively tighten lower and upper bounds on the bin loads based on an upper bound on the objective function value. The modified load domains can then be used to make further inferences, like the identification of non-packable bins (Shaw, 2004).
– *Dynamic capacity adjustment* Suppose that one knows the maximum load among all feasible sets that can still be packed starting from the current node, that is, after the latest extension. If that quantity is less than $\mathcal{C}$, the bin capacities for $k \geq \hat{k} + 2$ can be temporarily (locally) set to that value. This allows to potentially further tighten lower bounds and to increase the effectiveness of constraint propagation. The implementation of this technique is based on a bit array indicating for each feasible set whether it can currently be packed or not. For example, a feasible set that contains a chain that is already packed can be omitted from consideration. The bit array is indexed based on unique IDs that are assigned to the feasible sets upon creation at the root node and is then continuously updated upon branching (whenever a chain has been completely packed) and simply reverted upon backtracking. The update itself can be performed efficiently by relying on a hash table that allows to retrieve all feasible sets in which a particular chain is included.

7.4 Handling instances with small slices only

Instances that exclusively contain small slices, that is when $w_i \leq 0.5\mathcal{C}$ are more likely to fill bins exactly up to their load limit, which is the ideal constellation with regard to total weighted completion time minimization. Preliminary experiments have shown that this particularly applies to chains with $w_i \in [0, 0.3\mathcal{C}]$. This observation can be leveraged in so far that one might first try to obtain a solution with a maximum number of completely filled bins. If all bins but one (for the leftover slices) can be filled that way, it is clear that an optimal solution has been found.

The B&B algorithm, on the other hand, is designed to enumerate feasible sets, including those that do *not* completely fill a bin (see Sect. 6.3). The idea is to enhance the algorithm by a pre-optimization phase in the spirit of primal heuristics known from mixed integer programming. Instead of feasible sets with varying total load, only those that exactly cover a bin are generated. This can be achieved by solving a corresponding *subset sum* problem for each bin successively, starting with the first one. Of course, the precedence constraints exacerbate this process because they may restrict the opportunities for finding exact covers in later bins.

To find the desired packing, it might therefore be necessary to look at more than one covering feasible set for each bin k. Speaking in terms of the subset sum problem, one is interested in all or at least a subset of feasible sets whose total loads are equal to the capacity limit $\mathcal{C}$. In essence, this means that multiple (or even all) optimal solutions to a subset sum problem have to be enumerated. The well-known dynamic programming approach for this kind of problem has been modified to fill a table-like memory structure, allowing for an enumeration of equivalent solutions.

The subset-sum solution generator is then embedded into the B&B framework shown in Algorithm 1, replacing the "standard" feasible set generator and omitting filtering and bounding mechanisms. Instead, any branch that is not able to completely fill a bin (except for the last one) is immediately discarded. The number of branches is limited up front, rendering the approach similar to what is known as *beam search*. The idea is to run the algorithm with an increasing sequence of different "aperture" sizes, trying to dive as quickly as possible into the search tree. This pre-optimization phase is either terminated by a time limit or as soon as an optimal solution is found.

7.5 Generalizability considerations

So far, the B&B algorithm and its subroutines, based on concepts from Sects. 5 and 6, were described solely in the context

of a special case of total weighted completion time minimization. Although the resulting approach is able to optimally solve also real-world problem instances to optimality, as will be shown in Sect. 8, the applicability might appear limited with regard to the objective function. Instead, the proposed concepts are transferable to other objectives as well and thus broaden the scope of the overall approach. The chain-like precedence constraints are assumed to be still present for the subsequent considerations, since the exploitation of those constraints is the key to the efficacy and also the novelty of all the techniques.

Consider for example the objective of makespan minimization. Speaking in terms of packing, this coincides with the minimization of the number of allocated bins. All the symmetry breaking and dominance rules still hold under this objective as well. As far as the bound tightening approach is concerned, it gets immediately clear that the fundamental principle of approximating the compulsory part of the continuation of a partial packing is independent of the objective function. It is trivial to show that the approximation is a lower bound on the number of allocated bins. The tightening approach finally has to be embedded into lower bounds known from standard bin packing.

Lifting the restriction concerning the weights, hence allowing the sizes and weights of the slices to be independent but still homogeneous within the same chain, makes it possible to consider the TWCT objective in a more general fashion. A well-known special case is the minimization of the total completion time, with all weights $w_i = 1$. Though both cases (arbitrary weights and all weights equal to one) can in fact be handled by the B&B algorithm, the linkage to packing is not as close as for the variant considered in this paper. Indeed, trying to integrate the total completion time or the general TWCT objectives with the packing-oriented linear usage cost lower bounds (Braune, 2019) does not prove effective, as preliminary experiments have shown. The development of tailored lower bounds thus seems to be indispensable in this context.

8 Computational results

A series of computational experiments were conducted to empirically assess the performance of the proposed Branch-and-Bound algorithm for the discrete malleable task scheduling problem in comparison to "off-the-shelf" commercial MIP and CP solvers. The following subsections first describe the experimental conditions under which the study took place and then discuss the obtained results in detail. Besides the main optimization objective, that is, the minimization of the total weighted completion time (TWCT), an alternative objective, namely the minimization of the makespan (and thus the number of occupied bins) is addressed as well. This should point out the versatility of the proposed base concepts.

Table 3 Instance configurations for different problem sizes

Chains	Slices/chain	Slices total
{10, 20}	[3, 8], [2, 4]	50
{10, 20, 30}	[2, 4], [3, 8], [6, 15]	100
{10, 20, 30, 40}	[2, 5], [3, 6], [5, 10], [10, 20]	150
{10, 20}	[8, 13], [15, 25]	200
{10, 20}	[13, 18], [25, 35]	300

8.1 Experimental setup

The computational experiments are based on three types of problem data: problem instances generated randomly from scratch, benchmark sets from the literature, and finally real-world data sets. As for the newly generated instances, the problem size varies between 50 and 300 slices in total. Table 3 provides an overview of the configurations for each size. The number of chains ranges from 10 to 40 (first column). The chains have different lengths, with the respective ranges for the number of slices per chain shown in the second column. For the two largest instance groups with 200 and 300 slices, the number of chains is limited to 20. For each problem instance, five different capacity configurations ($\mathcal{C} \in \{20, 50, 100, 250, 500\}$) are used.

A further important distinctive feature is the range of weights. It is known for the problem variant without precedence constraints that the "bulkiness" of items has a considerable impact on the required computation time (Braune, 2019). Therefore, three different basic scenarios have been considered for the computational experiments: (9) a small-weight scenario in two variants, with $w_i \in [0, 0.3\mathcal{C}]$ and $w_i \in [0, 0.5\mathcal{C}]$, (10) instances with bulky slices, that is, $w_i \in [0.3\mathcal{C}, 0.7\mathcal{C}]$ and $w_i \in [0.2\mathcal{C}, 0.8\mathcal{C}]$, and (3) a mixed scenario, with $w_i \in [0, \mathcal{C}]$. For each configuration obtained this way, a set of 50 instances was generated.

From a structural point of view, instances of high-multiplicity bin packing problems would be best suited for benchmarking purposes. However, to the best of the author's knowledge, no such instances are publicly available. Therefore, the second part of the computational experiments is based on standard benchmark instances known from bin packing. They offer at least a certain degree of item multiplicity and can thus be subdivided into chains in a simple and reproducible way. The data sets taken from the OR-library include 120, 250 and 500 items (= slices) per instance. For two further groups of bin packing instances, `bin1data` and `bin2data`, as described by Scholl et al. (1997), $\mathcal{C} \in \{100, 120, 150, 1000\}$ and $\sum_i n_i = 50$, 100, 200, and 500.

Finally, the randomized instances used in the survey by Delorme et al. (2016) are also employed for assessing the efficacy of the proposed B&B approach. These instance sets cover four different problem sizes with 50, 100, 200, and 300 slices and are thus referred to as `DIM 50` through `DIM 300` henceforth.

All benchmark instances underwent a scaling procedure to reduce the amount of potential feasible sets. The instances are designed for classical bin packing, where much more effective filtering mechanisms can be used. Without these, none of the methods analyzed in this paper are able to achieve optimal solutions within a reasonable time frame, as preliminary experiments revealed. Hence, a scaling factor $\varphi \in]0, 1[$ was applied to both, weights and capacities in the following fashion: $w_i \leftarrow \lceil \varphi \cdot w_i \rceil$ and $\mathcal{C} \leftarrow \lceil \varphi \cdot \mathcal{C} \rceil$. The factor φ was set to 0.2 for an original capacity $\mathcal{C} \leq 200$ and 0.1 otherwise. These modified instances are available for download at https://bda.univie.ac.at/research/data-and-instances/scheduling-problems/discrmalleablesched/, besides the random data sets.

Apart from that, the proposed B&B method and its competitors were also applied to real-world instances, retrieved from a petrochemical R&D facility during a scheduling optimization project. In total, 10 instance sets, collected over 5 months, in intervals of two weeks were used for the experiments. Each set contains around 30 instances. The capacities are aggregated daily resource capacities given in minutes, ranging between 24 and 8280. The number of slices varies between five and 1281, yielding an average of 249 slices per instance. The instances cover human resources only and individual capacities (regular working times) are aggregated to the workgroup or team level.

The B&B algorithm described in Sect. 7 was implemented in C# and run on the Microsoft .NET platform. The configuration settings are summarized in Table 4. For comparison purposes, the MIP model from Sect. 4.2 and two constraint programming (CP) models described in Sects. 1 and 1 were implemented in IBM ILOG CPLEX and IBM ILOG CP Optimizer (version 12.10), respectively. The modifications required to cover the makespan objective are indicated in the corresponding subsection (Sect. 8.4). All experiments were conducted on a Windows 10 (× 64) workstation, equipped with a Core i7-4770 CPU (3.4 GHz) and 16 GB of RAM. All solver algorithms were run in single-threaded mode, given a time limit of 1800 s per instance.

Table 4 B&B configuration settings

Configuration property	Actual setting
Search strategy	Depth-first
Branching type	Feasible set-based
Branching order	(1) Total load (non-increasing)
	(2) # of slices (non-decreasing)
Lower bound	L_{2LLM}^{LUC} (tightened)
Pre-optimization phase time share	5 %

8.2 Generating initial incumbents

Although the precedence constraints of the problem at hand are relatively difficult to handle for an exact optimization method, as underpinned by various anomalies (see Sect. 6), they can be easily incorporated into existing construction heuristics known from bin packing. In fact, the only necessary modification is to keep track of any started chain to ensure its non-preemptive continuation. While this is easy to cover from an implementation point of view, it makes it more difficult to find good solutions, because each "bad" packing decision may immediately propagate across several subsequent bins. First fit decreasing (FFD) and best fit decreasing (BFD) variants are employed, besides a knapsack-based heuristic. The latter tries to maximize the allocated loads in each bin, proving particularly effective in problem settings with many small slices, as these are more likely to produce completely filled bins. A detailed description of these heuristics, including a pseudo-code notation, is given in Sect. 22. All heuristics were enhanced to properly cover the chain-like precedence constraints of the problem at hand.

A slight variation of the profits in the knapsack-based heuristic allows to keep the number of slices as small as possible at the same time: when subtracting one from each slice's profit ($g_{ij} \leftarrow g_{ij} - 1$), knapsacks with a larger number of slices are less preferable from a profit-oriented point of view. Tables 12 and 13 summarize heuristic results for the randomized instances, for the TWCT and the makespan objective, respectively. The knapsack-based algorithm aiming at minimizing the number of slices is referred to as "Knapsack$^{\text{Min}}$" in the table. For the TWCT objective, the table also reports results obtained by heuristics working according to the first- and best-fit increasing principle.

It turns out that FFD and BFD perform well in the makespan case, while their performance considerably declines for scenarios with bulky items under the TWCT objective. As expected, the knapsack-based heuristics perform exceptionally well as long as slices are small. In contrast, difficulties arise in the bulky or mixed settings.

To generate an initial incumbent solution for the B&B algorithm, all four heuristics are applied to the problem instance and the best resulting solution is installed. When comparing methods, the same solution is provided to the MIP and CP solvers as a "warm-start" (starting point) solution.

8.3 Total weighted completion time objective (proportional weights)

8.3.1 Results on randomized instances

As outlined in Sect. 8.1, the randomized instances have been designed to cover three levels of bulkiness. During the computational experiments, it turned out that this is in fact the most distinctive feature with regard to solvability and required computation time. The computational results reported in Table 5 are therefore grouped accordingly. The second grouping criterion is the number of chains (column "$|T|$"). For each combination of weight range and chain count, the results are aggregated over instance sets with $\mathcal{C}$ ranging from 20 to 500. Each such set contains 50 instances, yielding 250 instances per combination. For each of the compared approaches, the table lists the percentage of instances solved to optimality, and the average (t_μ), maximum ($t_{\max}$) and coefficient of variation (t_{CV}) of the computation times (in seconds). Only those instances for which an optimal solution could be found are included in the time statistics. The highest percentage of optimal solutions in each row is printed in bold face while the smallest average computation time is marked with a superscript "b". Note that the CP results reported here are based on the corresponding scheduling model described in Sect. 1. Though the performance of the CP models is generally poor for this kind of objective, the scheduling-oriented model still shows better results than the packing-based CP formulation (see Sect. 1).

Small slices with $w_i \in [0, 0.3\mathcal{C}]$ seem to pose the least difficulties among all settings. Thanks to the subset sum-based pre-optimization phase, the B&B is able to solve almost all of these instances to optimality within a fraction of a second. Strikingly, the percentage of optimal solutions achieved by the MIP actually increases with the number of chains $|T|$. It is obviously easier to find exact bin covers with a greater variety of chain sizes. A slight increase of the weight upper bound ($w_i \leq 0.5\mathcal{C}$) already changes the picture a little bit: The subset-sum phase is much less effective here, also for a higher number of chains. On the other hand, slices are too small for bounding techniques to work properly. It turns out that this setting is the hardest for the B&B when compared to the MIP. For $|T| = 40$, the ratio of optimally solved instances even becomes equal. There is also no clear winner in terms of average computation times.

Bulky slices ($w_i \in [0.3\mathcal{C}, 0.7\mathcal{C}]$ and $w_i \in [0.2\mathcal{C}, 0.8\mathcal{C}]$) basically reduce the combinatorics, because the probability that slices of different chains are just not compatible with each other is higher. On the other hand, exact bin covers are less likely, rendering the subset sum-based pre-optimization phase ineffective. Nevertheless, the B&B method performs well also in this scenario, leading to computation time advantages (average and maximum) of about an order of magnitude, as long as $|T| \leq 20$. More chains (≥ 30) make it considerably more difficult for both, the B&B and the MIP, to solve those instances.

As far as the mixed instances ($w_i \in [0, \mathcal{C}]$) are concerned, the B&B is still able to solve a large portion of instances to optimality, being between 5 and 60 times faster than the MIP on average.

The fact that none of the 40 chains instances in the bulky and mixed scenarios could be solved to optimality within the time limit underpins the difficulty of the problem. The coefficient of variation is fluctuating for all methods, with the MIP showing slightly smaller values than the B&B. However, since the B&B solves a notably higher percentage of instances, this statistics is somewhat biased.

To gain further insight into the ability of the different methods to quickly deliver optimal solutions, an investigation has been conducted that focuses on the number of such solutions obtained over time. Figure 10a and b plot the fraction of optimal solutions against the computation time for 10 and 20 chains, respectively. The total slice count is varied between 50 and 150 and $\mathcal{C} = 500$. The area shaded in light gray visualizes the range for the B&B method, while the area shaded in darker gray corresponds to the MIP results. Strikingly, those areas show almost no overlaps, meaning that the B&B is still able to deliver optimal solutions for 150 slices faster than the MIP is able to do it for 50 slices.

As a second aspect of the analysis, the impact of the total number of slices on the computation time is of central concern. This time, the weight range is kept fixed ($w_i \in [0, \mathcal{C}]$) and only the slice count is varied between 50 and 300. This is only possible because the number of chains $|T|$ has been limited to 20 for this kind of investigation. Figures 13 and 14 in 22 plot computation times and the percentage of instances solved to optimality against the total number of slices for 10 and 20 chains ($\mathcal{C} = 500$). In both scenarios, it can be observed that the computation times of the B&B are almost insensitive to the number of slices, while the number of achieved optimal solutions slightly decreases. In contrast, the MIP and CP approaches are more notably affected by the increase in the number of slices. This especially holds for the optimality statistics which begin to decline quite early. Note that no presentable results could be obtained for the CP approach in case $|T| = 20$ (see also Table 5).

The last part of the computational evaluation is concerned with the impact of the problem-specific enhancements to the B&B method, as presented in Sects. 5 and 6. To assess the contribution of each of those techniques with regard to B&B performance, the following approach has been adopted: The B&B is run on the mixed instance set ($w_i \in [0, \mathcal{C}]$, $\mathcal{C} = 500$) again, leaving out exactly *one* of the enhancements at a time. The required per-instance computation times are then compared to the original ones, obtained with the fully-fledged B&B. The comparison itself is based on time factors com-

Table 5 Randomized instances, TWCT objective: percentage of obtained optimal solutions and required computation times (seconds) across different methods

w_i	$\|T\|$	B&B				MIP				CP			
		% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}
$[0, 0.3\mathcal{C}]$	10	**98**	19^b	100	5.0	69	126	384	2.9	1	1,158	1,187	0.6
$[0, 0.3\mathcal{C}]$	20	**98**	0^b	4	4.6	76	30	80	4.0	0			
$[0, 0.3\mathcal{C}]$	30	**99**	0^b	1	4.9	82	34	126	3.6	0			
$[0, 0.3\mathcal{C}]$	40	**99**	0^b	1	4.9	84	63	236	4.5	0			
$[0, 0.5\mathcal{C}]$	10	**99**	41^b	218	2.2	32	389	1,387	1.8	14	580	672	0.9
$[0, 0.5\mathcal{C}]$	20	**67**	224	694	2.8	28	128^b	637	2.5	0			
$[0, 0.5\mathcal{C}]$	30	**38**	392	821	1.8	26	255^b	587	2.6	0			
$[0, 0.5\mathcal{C}]$	40	**26**	104^b	252	3.2	**26**	245	629	3.2	0			
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	10	**100**	0^b	0	1.8	71	386	799	1.3	46	824	1,677	0.7
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	20	**99**	36^b	175	2.1	34	613	1,655	1.0	0			
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	30	**46**	574^b	1,407	1.0	1	978	1,388	0.6	0			
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	40	0				0				0			
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	10	**100**	0^b	0	2.1	61	407	953	1.3	53	743	1,411	0.9
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	20	**98**	70^b	186	2.1	23	615	1,282	0.9	0			
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	30	**28**	617	734	0.9	0				0			
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	40	0				0				0			
$[0, \mathcal{C}]$	10	**100**	6^b	29	2.4	58	341	769	1.3	76	314	749	1.4
$[0, \mathcal{C}]$	20	**83**	186^b	460	1.5	17	956	1,613	0.8	0			
$[0, \mathcal{C}]$	30	**11**	783	1,090	0.8	0				0			
$[0, \mathcal{C}]$	40	0				0				0			

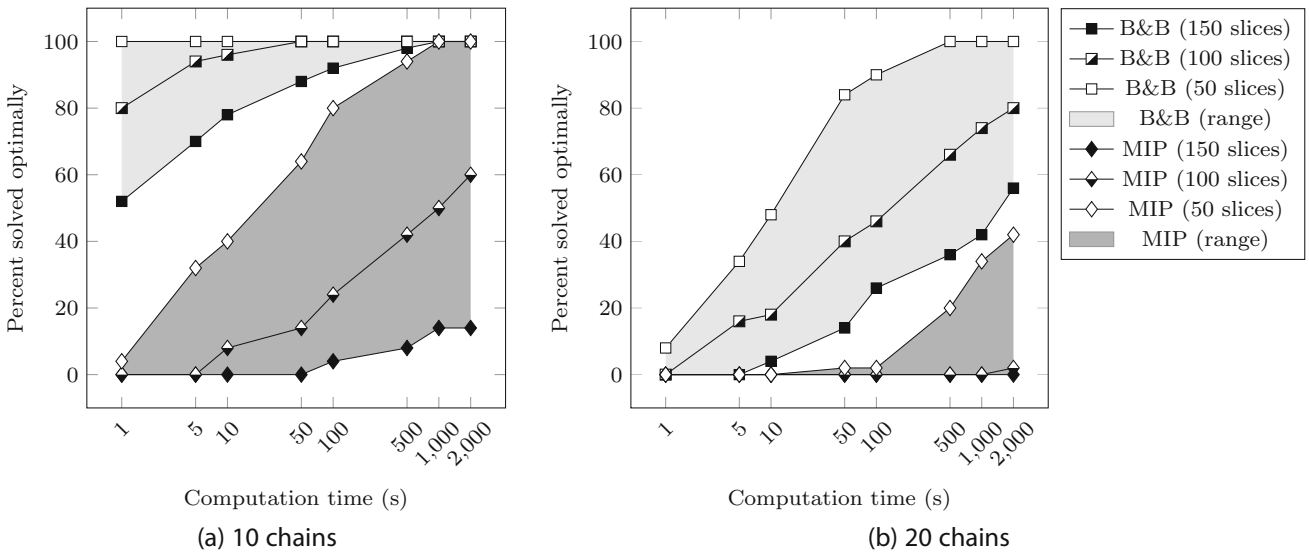

Fig. 10 Percentage of optimally solved instances over time ($w_i \in [0, \mathcal{C}]$ and $\mathcal{C} = 500$, TWCT objective)

puted as the quotient of the new computation time and the original one. An average time factor of 1.9, for example, means that the B&B algorithm without the enhancement in question is 1.9 times slower on average than the original B&B with all features activated. Table 6 reports the corresponding results for each enhancement in terms of average, minimum and maximum statistics. Besides that, the difference in the number of optimal solutions found is also reported as a percentage (column Δ_{Opt}). A value of -16, for example, means that without the approach in question, the proportion of opti-

Table 6 Impact of B&B ingredients ($w_i \in [0, \mathcal{C}]$, $\mathcal{C} = 500$, TWCT objective)

$\|T\|$	Slices	No no-goods				No tightening				No dom.rules			
		$\%\Delta_{Opt}$	Time factor			$\%\Delta_{Opt}$	Time factor			$\%\Delta_{Opt}$	Time factor		
			Avg.	Min	Max		Avg.	Min	Max		Avg.	Min	Max
10	50	0	1.9	0.7	9.2	0	1.8	1.0	4.9	0	2.5	1.0	11.3
10	100	0	10.4	0.8	189.2	0	2.7	1.0	6.5	0	2.9	1.0	13.8
10	150	−16	20.2	0.8	131.7	0	2.7	1.0	5.0	−2	3.1	1.6	11.0
20	50	−2	21.3	2.3	334.2	0	1.8	1.6	1.9	0	3.3	1.3	20.8
20	100	−34	9.1	2.0	30.8	−6	2.2	1.8	2.6	−10	4.5	1.7	24.5
20	150	−26	11.6	2.4	45.0	−18	2.3	2.0	2.9	−22	7.0	2.0	25.8
30	100	−8				−4	1.9	1.8	2.0	−6	2.2	2.2	2.2
30	150	−6				−4	1.9	1.9	1.9	−6			

mal solutions is 16% smaller. Note that the dominance rules described in Sects. 6.2 and 6.3 are considered in a combined fashion.

Obviously, the no-good recording technique based on equivalent packings (see Sect. 6.1) has the largest impact, rendering the B&B up to 21.3 times faster on average. It must be emphasized that none of the larger instances with 30 chains can be solved to optimality without no-good recording. Hence, this kind of enhancement appears to be vital to the performance of the B&B. Bound tightening seems to perform best under a small number of chains, while the dominance rules based on equally sized chains and extensible feasible sets show their best results for $|T| = 20$. Both of the latter two kinds of techniques behave similarly for the larger instances with 30 chains. From this empirical investigation, it can be concluded that each of the enhancements on its own has a non-negligible, if not crucial impact on the B&B solution behavior.

8.3.2 Results on benchmark instances

Table 7 shows aggregated results on benchmark instances from the literature. Again, the proposed B&B algorithm is compared to the MIP and the CP formulation, using the same configuration and time limit as for the randomized instances. For each group, the table reports the average, the maximum and the coefficient of variation of the computation times needed for the instances that can be solved to optimality. The first column of each method shows the percentage of optimal solutions. The highlighting follows the same principle as in Table 5.

It can be seen that the B&B performs particularly well on the first part (`bin1data`) of the benchmarks according to (Scholl et al., 1997). The same holds true for the OR-library instances, none of which could be solved to optimality by the MIP or the CP. The `bin2data` sets are obviously harder to tackle. The ratio of optimal solutions is still relatively high for the B&B, while the MIP solver exhibits lower computation times for two subsets (N2 and N4). Anyway, the corresponding percentage of optimal solutions is considerably lower for both of them.

As regards the instances by Delorme et al. (2016), the solution behavior of the three approaches is similar to the `bin1data` instances. The B&B algorithm is able to solve between 70 and 90 percent of the instances, taking considerably less time than MIP and CP. Interestingly, the CP formulation performs a little bit better on these instances, which is contrary to the other instance sets from the literature.

8.3.3 Results on real-world instances

From Table 8, it becomes apparent that most instances drawn from real-world data are not very difficult to solve. In most of the cases, the slices are relatively small, rendering the construction of ideal packings quite easy. The B&B can solve between 80 and 90% of all instances to optimality, with average computation times between less than one and 30 s. Both MIP and CP take considerably more time (up to 3 min on average and almost half an hour maximum), with optimality ratios ranging between 63 and 75% for the MIP and between 38 and 50% for the CP.

8.4 Makespan objective

Minimizing the makespan for the given scheduling problem essentially means to allocate the smallest possible number of bins for the slices. For this objective, it is possible to rely on almost the same B&B configuration as for the TWCT objective, with the following differences: (1) bounding is based on the lower bound L_{2LLM} (Labbé et al., 1991), known from standard bin packing, and (2) no constraint propagation procedure is used. It is easy to verify that the symmetry breaking and dominance rules proposed in Sect. 6 are also valid in the makespan case. For the MIP and CP models stated in Sects. 4.2 and 1, just the objective function has to be adapted

Table 7 Benchmark instances from the literature, TWCT objective: ratio of obtained optimal solutions and required computation times across different methods

Instance set	B&B				MIP				CP			
	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}
bin1data (N1)	**100**	0^b	6	2.1	66	339	1,773	1.2	12	826	1,704	0.6
bin1data (N2)	**100**	7^b	148	2.7	6	537	1,474	0.8	0			
bin1data (N3)	**98**	90^b	1,485	2.2	0				0			
bin1data (N4)	**69**	154^b	1,730	2.1	0				0			
bin2data (N1)	**97**	33^b	1,607	5.4	75	54	1,435	4.1	1	1,764	1,764	
bin2data (N2)	**84**	48	1,629	5.0	53	20^b	411	3.6	0			
bin2data (N3)	**73**	15^b	231	2.8	33	34	653	4.2	0			
bin2data (N4)	**61**	112	1,414	2.4	23	15^b	37	0.5	0			
OR-library (120)	**100**	4	17	1.1	0				0			
OR-library (250)	**100**	42	212	1.1	0				0			
OR-library (500)	**90**	718	1,768	0.6	0				0			
DIM 50	**91**	34^b	1,076	3.7	43	338	1,784	1.4	27	211	1,787	1.8
DIM 100	**81**	42^b	1,315	3.3	13	467	1,714	1.0	16	196	1,795	1.8
DIM 200	**75**	73^b	1,377	2.7	0				10	384	1,696	1.3
DIM 300	**70**	85^b	1,366	2.4	0				7	330	1,450	1.0

Table 8 Real-world instances, TWCT objective: ratio of obtained optimal solutions and required computation times across different methods

Instance set	B&B				MIP				CP			
	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}
1	**84**	31^b	822	5.1	63	104	1,368	3.0	50	134	926	2.2
2	**82**	5^b	144	5.1	65	32	340	2.6	47	99	911	2.4
3	**85**	1^b	21	3.7	71	69	550	2.2	41	9	79	2.4
4	**88**	3^b	71	5.1	75	176	1,679	2.3	38	153	1,720	3.2
5	**82**	0^b	1	4.6	70	12	105	2.3	39	19	226	3.2
6	**78**	0^b	1	3.3	75	100	1,084	2.7	41	31	323	2.9
7	**87**	0^b	2	3.0	74	13	132	2.2	45	53	286	1.8
8	**84**	0^b	5	3.1	72	11	62	1.5	38	11	66	1.9
9	**88**	9^b	235	4.8	67	55	1,076	4.1	45	18	214	3.1
10	**84**	13^b	181	3.6	72	53	405	2.1	41	6	38	1.8

accordingly. Apart from a pure performance comparison, a further goal of the computational analysis in this context was to investigate in how far the discrete malleable scheduling problem with makespan objective differs from the standard bin packing problem. For this purpose, the well-known and publicly available solver of Brandao and Pedroso (2016) was employed. According to the survey of Delorme et al. (2016), it is one of the most efficient solvers in this field. Consequently, it was possible to solve all randomized instances to optimality. However, the precedence constraints are not considered by the solver and therefore, a major aspect of the analysis is to see how many of those precedence constraints are actually violated in the generated solutions.

Table 9 shows computational results obtained for the same randomized instances as used for the TWCT experiments. The comparison between the three solution approaches reveals that the B&B is still able to achieve the highest ratio of optimal solutions in most of the settings and is still faster in many cases. However, the CP model performs very well under this objective, especially for the instances with smaller slices. Unsurprisingly, the `pack` constraint (Shaw, 2004) blends much better with makespan minimization than with the sum-based TWCT objective, even when precedence constraints are imposed. As a consequence, the advantage of the B&B algorithm is much smaller, but a custom implementation of the `pack` constraint can be expected to considerably boost its performance, especially for instances with a greater number of chains.

The comparison to standard bin packing solutions is conducted in the last two columns of Table 9. Column "Δ bins"

Table 9 Randomized instances, makespan objective: percentage of obtained optimal solutions, required computation times (seconds) across different methods, and deviations from standard bin packing solutions

w_i	$\|T\|$	B&B				MIP				CP				VP Solver	
		% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}	Δ bins	Viol.
$[0, 0.3\mathcal{C}]$	10	**100**	1^b	9	6.2	97	1	9	0.7	**100**	8	57	3.0	0.0022	8.2
$[0, 0.3\mathcal{C}]$	20	99	7	54	6.7	98	4	34	1.5	**100**	5^b	38	2.2	0.0067	19.4
$[0, 0.3\mathcal{C}]$	30	99	0^b	0	7.0	99	0	0	0.2	**100**	0^b	2	2.4	0.0050	33.2
$[0, 0.3\mathcal{C}]$	40	99	0^b	0	7.0	99	0	0	0.1	**100**	1	4	1.5	0.0200	43.8
$[0, 0.5\mathcal{C}]$	10	**100**	12^b	68	4.1	64	68	244	3.4	84	75	301	3.8	0.0222	7.5
$[0, 0.5\mathcal{C}]$	20	**90**	67	210	3.7	65	37	138	3.6	81	64^b	199	3.8	0.1144	16.9
$[0, 0.5\mathcal{C}]$	30	78	96	194	4.0	70	23	111	2.4	**80**	41^b	110	2.9	0.2267	27.4
$[0, 0.5\mathcal{C}]$	40	62	29^b	76	5.4	62	39	157	1.2	**70**	30	96	3.1	0.4700	36.9
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	10	**100**	0^b	1	2.9	79	326	812	1.4	60	50	130	3.8	0.0000	2.4
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	20	**94**	74	224	3.2	61	377	930	1.6	56	31^b	118	4.1	0.0033	3.5
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	30	**64**	7^b	43	4.3	24	381	736	1.1	40	19	89	3.7	0.0833	5.0
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	40	**68**	5^b	25	2.2	6	558	998	1.1	45	5^b	10	2.8	0.1133	5.7
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	10	**100**	0^b	1	3.3	70	345	826	1.5	59	37	122	3.5	0.0089	2.7
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	20	**94**	66	139	3.0	54	421	980	1.3	59	24^b	88	3.8	0.0133	4.6
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	30	**54**	58	241	3.7	14	579	955	0.9	46	49^b	173	2.6	0.3233	7.1
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	40	**61**	17^b	72	2.5	5	533	906	1.0	50	40	101	2.9	0.3900	8.7
$[0, \mathcal{C}]$	10	**100**	5^b	22	3.9	68	317	863	1.6	66	42	102	3.6	0.0200	3.3
$[0, \mathcal{C}]$	20	**93**	46	112	2.9	49	334	644	1.5	63	15^b	46	3.7	0.0211	6.6
$[0, \mathcal{C}]$	30	**59**	42	119	3.6	16	353	465	1.3	51	32^b	78	3.3	0.2983	11.1
$[0, \mathcal{C}]$	40	**55**	35	76	3.3	7	299	598	0.7	50	23^b	54	3.3	0.6267	14.7

shows the mean difference in the number of bins allocated by the best B&B solution (optimal or feasible) and Brandao's VP solver. Column "Viol." indicates the average number of precedence constraints violated by the VP solver. It can be observed that the difference in the number of allocated bins is very low across all configurations, whereas the number of constraint violations is fluctuating a lot. It has to be remarked that the violations are counted based on the solution that the VP solver returns as "the optimal" solution for each instance. Clearly, there might be multiple optimal solutions for one and the same instance, with varying numbers of violated constraints. However, trying to reduce the violations on a per-instance basis would suggest an enumeration of optimal solutions, which in turn requires modifications to the VP solver and much longer computation times.

Obviously, most violations are produced for instances with smaller slices, and the number increases with the chain count. On the other hand, instances with bulky slices lead to very few precedence violations. For the mixed instances ($w_i \in [0, \mathcal{C}]$), the extent of the violations is still non-negligible, and therefore it can be concluded that solutions for the standard bin packing problem are very similar in terms of the number of allocated bins, but can hardly be considered as an alternative for the discrete malleable scheduling problem at hand.

Computational results on benchmark instances from the literature and on the real-world instances are summarized in Sect. 22 (Tables 10 and 11). Finally, a comparison of a variety of simple construction heuristics under the makespan objective is given in Table 13 (Appendix G) for the randomized instances.

9 Conclusion and outlook

An exact solution method has been presented for solving a multiprocessor scheduling problem in which the tasks are subject to a restricted form of discrete malleability and the objective is to minimize the total weighted completion time. Delayed precedence constraints of min/max type are used to concatenate unit-time task slices which suggest adopting a bin packing-oriented point of view. In fact, a packing-based branch-and-bound algorithm is proposed that makes use of several non-trivial, problem-specific enhancements. These include on the one hand a tightening approach for lower bounds that takes advantage of the special kind of precedence constraints. On the other hand, symmetry breaking and dominance rules have been devised to help reduce the size of the search tree. Since also non-maximal feasible sets of task slices have to be taken into consideration for computing optimal solutions to the problem at hand, these enhancements

to the B&B method turn out to be of crucial importance, as confirmed by computational experience. As a side track to the main solution approach, a pre-optimization phase has been proposed, based on a limited enumeration of subset-sum solutions. This kind of primal heuristic specifically aims at scenarios with small slices and thus a high probability of completely (and exactly) using up the capacity in each time period.

Due to a lack of benchmark sets in related multiprocessor literature, the computational evaluation is based on dedicated randomized instances, adapted benchmark problems from classical bin packing and data taken from a real-world setting. A custom MIP and a scheduling-oriented CP formulation serve as the comparison baseline. It must be pointed out again that the problem structure prohibits a simple adaptation or a direct transfer of existing sophisticated solution approaches (like column generation) from both fields, scheduling and packing.

In all the scenarios, the new B&B algorithm proves very effective in obtaining optimal solutions within the predefined time limit. The required computation times are more than an order of magnitude smaller than those needed by the MIP in the majority of cases. Whenever the average computation time of the MIP is smaller, its percentage of achieved optimal solutions is also notably lower. Surprisingly, the CP solver was not able to yield competitive results, most likely because of the structure (min-sum) of the objective function and the special type of precedence constraints. The good overall performance of the B&B method is mainly owed to the symmetry breaking and dominance rules, as well as the bound tightening approach, as an in-depth analysis reveals.

Instances with slices requiring between zero and half the number of resource units still seem to pose a challenge to the new method. Potential reasons might be the insufficient lower bound quality for this size configuration (Braune, 2019), and the inability of the primal heuristic to take effect. The real-world data in its current form mainly consists of slices that are small compared to the processor capacity. Thanks to the primal heuristic, this kind of problem can be handled quite well, and still better than through the MIP formulation.

The sample application to makespan minimization as an alternative objective gives an indication of the generalizability of the proposed concepts. Even without an appropriate constraint propagation mechanism, the B&B algorithm can still solve more instances than the MIP and CP models, but the advantage is notably smaller.

Gaining deeper insights into the real-world setting will hence be one of the key aspects of future research activities. On the theoretical side, there is still room for improvement with regard to feasible set combinatorics. The number of such sets to be created especially in the root node is still enormous for 40 or more chains. The development of even more effective elimination techniques or dominance rules will be of central concern in this context. Yet another stream of research might focus on the generalization of the proposed techniques and the overall optimization approach toward size-independent weights and/or arbitrary minimum and maximum time lags.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

Funding Open access funding provided by University of Vienna.

Appendix A: A packing-based CP model

Using custom constraints, a packing-based constraint programming formulation can be given for the problem considered in the paper in a clear and also very compact fashion. Let γ denote the vector of bin loads, hence γ_k represents the load of bin k, with $1 \leq k \leq \bar{k}$ and $\gamma_k \in \{0, \ldots, C\}$. The decision variable k^* reflects the number of bins used by the packing solution. The CP packing formulation can be stated in the following compact form:

$$\min \quad \sum_{k=1}^{\bar{k}} k \cdot \gamma_k$$

$$\texttt{pack}(\gamma, s, w, k^*), \tag{19}$$

$$s_{ij} \leq s_{i,j+1} \quad \forall\, i \in T,\ \forall\, 1 \leq j \leq n_i - 1, \tag{20}$$

$$s_{i,j+1} \leq s_{ij} + 1 \quad \forall\, i \in T,\ \forall\, 1 \leq j \leq n_i - 1. \tag{21}$$

The pack-constraint (Constraint 19) takes bin loads γ, bin assignment variables s (one for each item/slice), item sizes w and the number of used bins k^* as arguments. Apart from w, these are all decision variables whose values will be set by the solver. Note that the bin assignment s_{ij} of slice (i, j) can be seen as its starting time (with a constant offset of one, see Sect. 2) and hence the same symbol is used. The precedence constraints are more naturally represented in the conventional fashion here (see Constraints (20) and (21)), since the bin assignment variables are of type integer. Note

that the bin capacity constraints are indirectly enforced via the domains of decision variables γ_k.

Appendix B: A scheduling-oriented CP model

Constraint programming (CP) solvers like IBM ILOG CP Optimizer offer a broad variety of specific scheduling-related modeling facilities. For many scheduling problems, these dedicated model elements (mostly predefined custom propagators) can considerably accelerate the solution process, both in terms of feasibility and optimization. Therefore, as an alternative to the packing model from the preceding section, a scheduling-oriented CP formulation is introduced below, relying on unit-time interval variables Υ_{ij}, one for each slice (i, j):

$$\min \quad \sum_{i \in T} \sum_{j=1}^{n_i} size_i \cdot \texttt{endOf}(\Upsilon_{ij}) \tag{22}$$

$$\text{s.t.} \quad \sum_{i \in T} \sum_{j=1}^{n_i} \texttt{pulse}(\Upsilon_{ij}, size_i) \leq m, \tag{23}$$

$$\texttt{endBeforeStart}(\Upsilon_{ij}, \Upsilon_{i,j+1}, -1), \quad \forall i \in T, \; \forall 1 \leq j \leq n_i - 1, \tag{24}$$

$$\texttt{endBeforeStart}(\Upsilon_{i,j+1}, \Upsilon_{ij}, -2), \quad \forall i \in T, \; \forall 1 \leq j \leq n_i - 1. \tag{25}$$

The left-hand side of Constraint (23) represents a *cumul function expression*, speaking in terms of CP Optimizer. In the given context, it is used to limit the resource usage to the number of processors at any time. Constraints (24) and (25) ensure that the minimum and maximum delay precedence constraints are met. Note that the model is no more "time-indexed", being one of the most appealing properties of CP scheduling formulations in general.

Appendix C: Constraint propagation for bin packing with linear usage cost

Cambazard et al. (2013) proposed constraint propagation-based domain reduction rules for the a packing problem with linear usage cost. These rules are also embedded into the Branch-and-Bound algorithm for the problem addressed in this paper (see Sect. 7. For the sake of completeness, the associated procedures are described in detail below. Note that this presentation is essentially a reproduction of the original description, with slight notational modifications.

The central idea of the approach is to tighten the domains of the variables representing the bin loads, that is, their lower and upper bounds, denoted by $\underline{\gamma}_k$ and $\overline{\gamma}_k$, respectively. Let us assume that $\underline{\gamma}_k > 0$ for the first few bins, as indicated by the hatched area in Fig. 11. Similarly, let us suppose that $\overline{\gamma}_k < \mathcal{C}$ for all bins (light gray boxes in Fig. 11), implying that some tightening has already taken place. A lower bound on the objective value can be computed by fractionally packing all items into the bins, from left to right, filling each bin up to its upper bound $\overline{\gamma}_k$. The last bin that receives fractional load will be called the *boundary* bin henceforth, and is denoted by $\hat{k}$. The fractional load allocated to bin $\hat{k}$ might be smaller than $\overline{\gamma}_{\hat{k}}$.

Algorithm 2: Outline of the CP procedure for load bound adjustment, according to Cambazard et al. (2013).

Input : Bin load upper and lower bounds $\overline{\gamma}$ and $\underline{\gamma}$, respectively, an upper bound UB on the objective value, the total slice weight W, the index $\hat{k}$ of the boundary bin

Output: A boolean value, indicating whether the CP model is consistent or not

1 Set $W' \leftarrow W - \sum_{k=1}^{\hat{k}} \underline{\gamma}_k$;
2 Set $L_{\hat{k}} \leftarrow W' - \sum_{k=1}^{\hat{k}-1} (\overline{\gamma}_k - \underline{\gamma}_k)$;
3 Set $L_k \leftarrow \overline{\gamma}_k - \underline{\gamma}_k$, for all $1 \leq k < \hat{k}$;
4 Set $L_k \leftarrow 0$, for all $k > \hat{k}$;
5 $LB_1 \leftarrow \sum_{k=1}^{\hat{k}-1} L_k \cdot k + L_{\hat{k}} \cdot \hat{k}$;
6 $LB'_1 \leftarrow \sum_{k=1}^{\hat{k}} \underline{\gamma}_k \cdot k + LB_1$;
7 `UpdateLoadLowerBounds`$(\underline{\gamma}, UB, LB'_1, \hat{k}, L)$;
8 `UpdateLoadUpperBounds`$(\overline{\gamma}, UB, LB'_1, \hat{k}, L)$;
9 Check consistency based on non-packable bins;

Algorithm 3: Outline of procedure `UpdateLoadLowerBounds`.

Input : Lower bounds on bin loads ($\underline{\gamma}$), an upper bound and a lower bound on the objective value (UB and LB'_1), the index $\hat{k}$ of the boundary bin, and a vector L of fractional loads.

Output: Adjusted bin load lower bounds $\underline{\gamma}$

1 Set $gap \leftarrow UB - LB'_1$;
2 **for** $k \leftarrow 1$ **to** $\hat{k}$ **do**
3 **if** $k = \hat{k}$ **then** $b \leftarrow \hat{k} + 1$ **else** $b \leftarrow \hat{k}$ Set $q \leftarrow 0$, $\Delta_{Obj} \leftarrow 0$;
4 **while** $q < L_k$ *and* $\Delta_{Obj} \leq gap$ *and* $b \leq \bar{k}$ **do**
5 $q' \leftarrow \min(L_k - q, \overline{\gamma}_b - \underline{\gamma}_b - L_b)$;
6 $\Delta'_{Obj} \leftarrow q' \cdot (b - k)$;
7 **if** $\Delta_{Obj} + \Delta'_{Obj} > gap$ **then**
8 $q' \leftarrow (gap - \Delta_{Obj})/(b - k)$;
9 **end**
10 $q \leftarrow q + q'$;
11 $\Delta_{Obj} \leftarrow \Delta_{Obj} + \Delta'_{Obj}$;
12 $b \leftarrow b + 1$;
13 **end**
14 **if** $\Delta_{Obj} > gap$ **then** $\underline{\gamma}_k \leftarrow \underline{\gamma}_k + (L_k - q)$
15 **end**

When trying to increase the load lower bound of a bin $1 \leq k < \hat{k}$, one tries to determine the maximum load q that can be removed from bin k and placed into the earliest bin that still has capacity left, i.e., primarily bin $\hat{k}$, without exceeding the upper bound on the objective function value by that load transfer. In Fig. 11, the movable load is shown for bin 1 (q_1). When $\hat{k}$'s remaining capacity is not sufficient to accommodate q_k, the remaining part of the latter quantity is simply put into the next bin $\hat{k}+1$. When this one is depleted, one tries $\hat{k}+2$, and so on. If the upper bound is actually exceeded by moving a quantity $q_k < \overline{\gamma}_k - \underline{\gamma}_k$, the lower bound can be set to $\overline{\gamma}_k - q_k$.

On the other hand, when trying to adjust (= decrease) the bin load upper bounds $\overline{\gamma}_k$, for $\hat{k} \leq k \leq \bar{k}$, the idea is to take some (fractional) load q from the boundary bin $\hat{k}$, and try to move it to a particular bin k , without violating the objective value upper bound. Figure 12 illustrates this principle. In the shown case, it is assumed that no more than the quantity removed from bin $\hat{k}$ (q_{12}) can be placed into bin 14 without exceeding the upper bound on the objective value. This means that the upper bound of bin 14 could actually be reduced to q_{12}. Note that the movable quantity q might stretch across several bins b, with $1 \leq b \leq \hat{k}$, if the differences between lower and upper bounds are small in that area, and later bins still have ample space.

Algorithms 3 and 4 give an outline of the two procedures used for tightening lower and upper bounds, respectively. Both procedures rely on a vector L of fractional loads, defined for bins $1 \leq k \leq \hat{k}$. These loads originate from the fractional packing determined during bounding the objective function value from below. Since the load is not allow to drop below the respective lower bound, only the quantities L_k can be subject to a move to a later bin. The initialization of those quantities takes place in the main CP routine (see Algorithm 2) that also embeds the two tightening procedures. The final step of the propagation is a consistency check using the concept of non-packable bins according to Shaw (2004).

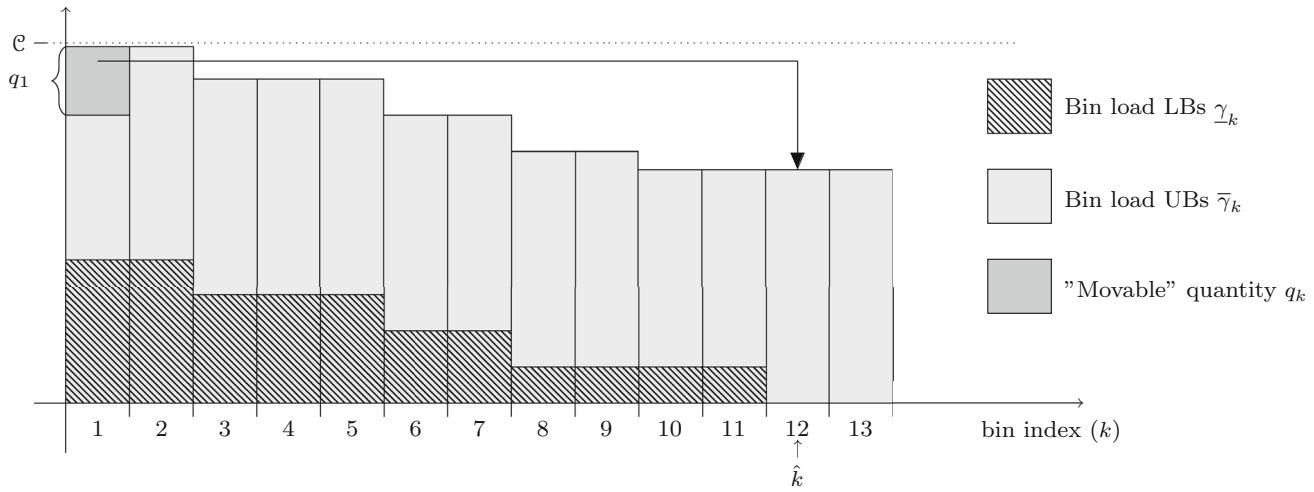

Fig. 11 Principle of tightening bin load lower bounds

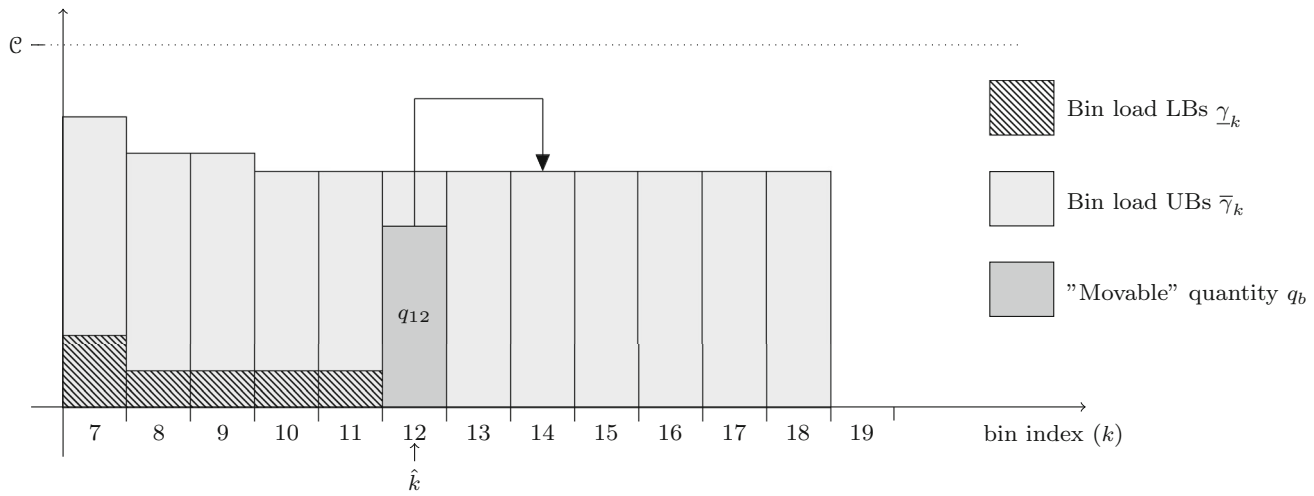

Fig. 12 Principle of tightening bin load upper bounds

Algorithm 4: Outline of procedure `UpdateLoadUpperBounds`.

Input : Upper bounds on bin loads ($\overline{\gamma}$), an upper bound and a lower bound on the objective value (UB and LB'_1), the index $\hat{k}$ of the boundary bin, and a vector L of fractional loads.

Output: Adjusted bin load upper bounds $\overline{\gamma}$

Set $gap \leftarrow UB - LB'_1$;
for $k \leftarrow \hat{k}$ **to** $\bar{k}$ **do**
  Set $q \leftarrow 0$, $\Delta_{Obj} \leftarrow 0$;
  Set $b \leftarrow \hat{k}$;
  if $j = \hat{k}$ **then**
    $b \leftarrow \hat{k} - 1$;
    $q \leftarrow L_{\hat{k}}$;
  end
  while $q < \overline{\gamma}_j - \underline{\gamma}_j$ *and* $\Delta_{Obj} \leq gap$ *and* $b \geq 1$ **do**
    $q' \leftarrow \min(L_b, \overline{\gamma}_j - \underline{\gamma}_j - q)$;
    $\Delta'_{Obj} \leftarrow q' \cdot (k - b)$;
    if $\Delta_{Obj} + \Delta'_{Obj} > gap$ **then**
      $q' \leftarrow (gap - \Delta_{Obj})/(k - b)$;
    end
    $q \leftarrow q + q'$;
    $\Delta_{Obj} \leftarrow \Delta_{Obj} + \Delta'_{Obj}$;
    $b \leftarrow b - 1$;
  end
  if $\Delta_{Obj} > gap$ **then** $\overline{\gamma} \leftarrow \underline{\gamma} + q$
end

Appendix D: Construction heuristics for the bin packing problem with chain-like precedence constraints

Construction heuristics, like first-fit decreasing (FFD) or best-fit decreasing (BFD) are well known from standard bin packing. In this section, it is shown how the chain-like precedence constraints can be integrated into the basic heuristics and a further approach that is based on solving a knapsack problem in every bin.

The idea of the integration into the fit-based algorithms is to process the slices in a chain-wise manner. Hence, a chain is considered a single entity to be scheduled, in a discrete malleable fashion, however. Algorithm 5 exemplarily shows the flow principle of a precedence constraint-aware fit-based algorithm. First, the chains are sorted according to their weight (= size of their slices) and the largest chain is selected to be packed starting with the first fitting bin. Before that, it has to be verified that the chain can be packed consecutively without any preemption. The latter would occur, for example, if one of the subsequent bins does not have enough capacity any more to accommodate at least one slice of the chain. Once a chain is released for packing, the procedure tries to put as many slices of this chain as possible into each bin in a row. This does not violate the FFD principle, because all these slices are of the same size. A best-fit decreasing (BFD) algorithm could be realized within the same structural framework, by slightly modifying the condition of the first while-loop such that the best fitting bin is sought instead of the first one.

When packing (scheduling) unit-time slices with $w_{ij} = size_{ij}$ and total weighted completion time objective, it is known that left-shifted packings are preferable (Braune, 2019). Hence, when packing the bins from left to right in chronological order, the leftmost bins should receive as much load as possible. An obvious approach to achieve this is to solve a knapsack problem for each bin, based on the remaining slices. Algorithm 6 gives an outline of the corresponding heuristic. First, the slice profits, denoted by g_{ij} are set equal to the weights. As long as there are chains with unpacked slices (set $\mathscr{R}$), the current bin k first receives exactly one slice of each chain that is in-progress to maintain the non-preemption property. For the remaining part of the bin, a knapsack problem is set up, involving all slices that could potentially be packed (set $\mathscr{U}$). The subroutine `SolveKnapsack` is not described in detail here, because it can be realized by any algorithm (exact or heuristic) that solves the knapsack problem. It just has to return the slices (set $\mathscr{K}$) that are actually placed into the knapsack and thus the remaining part of the bin.

Algorithm 5: First-fit decreasing (FFD) with chain-like precedence constraints.

Sort chains according to their slice size in non-increasing order;
Re-index the chains such that $i < j \implies w_i \geq w_j$;
for $i \leftarrow 1$ **to** $|T|$ **do**
  `/* Find first fitting bin */`
  $k \leftarrow 1$;
  while $k \leq \bar{k}$ *and* $\mathcal{C} - \gamma_k < w_i$ *and not the whole chain can be accommodated* **do**
    $k \leftarrow k + 1$;
  end
  `/* Pack chain starting from bin k */`
  $j \leftarrow 1$;
  while $k \leq \bar{k}$ *and* $j \leq n_i$ **do**
    Pack as many slices of chain i as possible into bin k;
    Increase slice index j accordingly;
    $k \leftarrow k + 1$;
  end
end

Algorithm 6: Knapsack-based packing with chain-like precedence constraints.

```
Set profits g_ij ← w_ij, for all i ∈ T, j ∈ {1, ..., n_i};
Init pointers to next slice per chain: u[i] ← 1, for all i ∈ T;
Set k ← 1;
Init set R ← {i | i ∈ T} of remaining chains;
while R ≠ ∅ do
    /* Pack in-progress chains first into bin
       k                                        */
    Init remaining capacity C' ← C;
    foreach i ∈ T do
        if chain i is in progress then
            Pack exactly one slice of chain i into bin k;
            Set pointer u[i] ← u[i] + 1;
            Set C' ← C' − w_i;
            if all slices of chain i are packed then
                R ← R \ {i};
            end
        end
    end
    /* Solve knapsack for rem. part of bin */
    Determine set U of unpacked slices:
    U ← {(i, j) | i ∈ T, u[i] ≤ j ≤ n_i};
    K ← SolveKnapsack(U, C', w, g);
    Pack all slices (i, j) ∈ K into current bin k;
    Update u and R;
    Set k ← k + 1;
end
```

Appendix E: Impact of total slice count on solution behavior

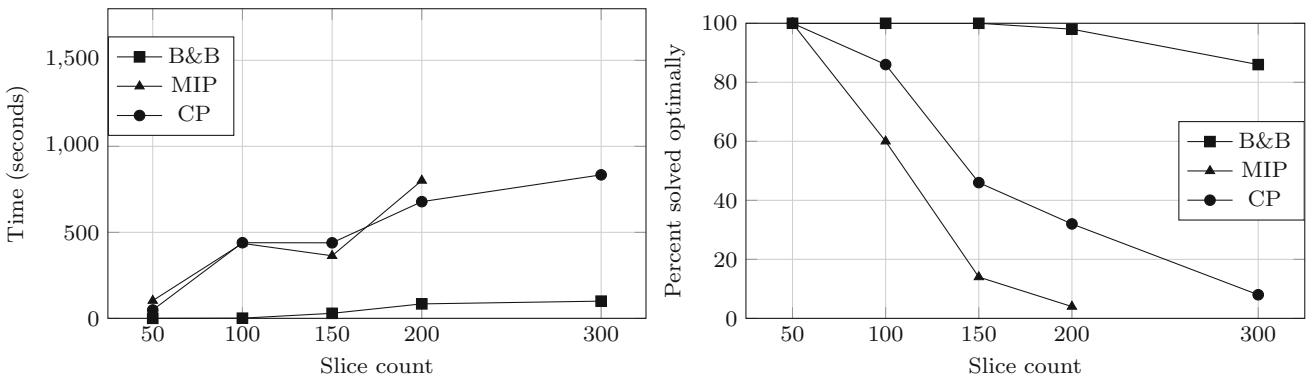

Fig. 13 Comparison of B&B, MIP and CP results for up to 300 slices (10 chains, $size_i \in [0, m]$ and $m = 500$)

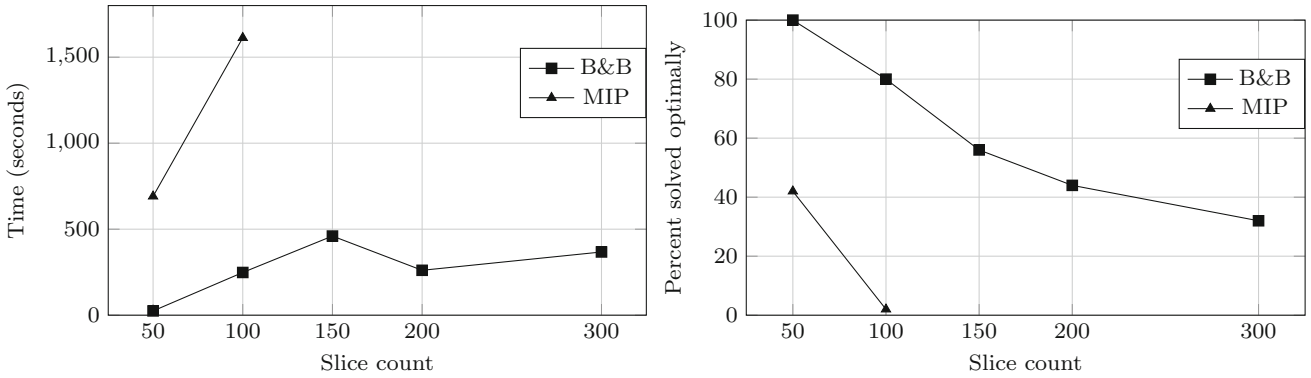

Fig. 14 Comparison of B&B, MIP and CP results for up to 300 slices (20 chains, $size_i \in [0, m]$ and $m = 500$)

Appendix F: Additional computational results for the makespan objective

Table 10 Benchmark instances from the literature, makespan objective: ratio of obtained optimal solutions and required computation times across different methods

Instance set	B&B				MIP				CP			
	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}
bin1data (N1)	**100**	0^b	14	6.6	83	196	1,761	2.0	91	4	291	8.1
bin1data (N2)	**100**	3^b	127	5.1	26	274	1,556	1.3	73	62	1,703	3.7
bin1data (N3)	**100**	11^b	232	2.7	0				54	171	1,734	2.0
bin1data (N4)	**91**	159	1,731	2.1	0				25	85^b	940	2.3
bin2data (N1)	**99**	10	573	6.0	91	34	1,129	5.1	97	6^b	230	4.7
bin2data (N2)	**93**	10	435	4.9	75	5^b	292	6.6	85	38	1,236	5.3
bin2data (N3)	**77**	41	1,056	4.2	63	2^b	6	0.6	67	34	1,374	5.7
bin2data (N4)	**60**	34	921	4.6	46	123	972	1.4	48	28^b	1,345	6.4
OR-library (120)	**100**	4^b	28	2.0	0				75	208	1,012	1.6
OR-library (250)	**100**	52^b	376	1.8	0				10	514	1,019	1.4
OR-library (500)	**75**	591	1,549	0.9	0				0			
DIM 50	**97**	15^b	1,330	7.6	79	155	1,697	2.2	89	17	1,619	6.9
DIM 100	**90**	24^b	1,340	5.0	20	593	1,795	0.9	71	51	1,205	3.5
DIM 200	**84**	52^b	1,426	3.6	0				43	122	1,777	2.4
DIM 300	**79**	61^b	1,405	3.2	0				33	183	1,730	2.1

Table 11 Real-world instances, $TWCT^p$ objective: ratio of obtained optimal solutions and required computation times across different methods

Instance set	B&B				MIP				CP			
	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}	% Opt	t_μ	$t_{\max}$	t_{CV}
1	**97**	6	115	4.0	81	84	903	2.4	88	2^b	46	5.2
2	**97**	0^b	1	3.6	91	117	1,569	2.8	91	49	1,525	5.6
3	**100**	0^b	1	4.8	88	29	310	2.1	97	9	289	5.7
4	**100**	8	268	5.6	91	82	518	1.8	94	7^b	214	5.5
5	**97**	0^b	4	5.4	88	73	1,429	3.6	97	2	57	5.6
6	**100**	0^b	0	4.4	91	91	929	2.5	100	19	613	5.7
7	**100**	0^b	1	4.1	94	36	298	2.0	97	0^b	0	3.0
8	**100**	0^b	1	4.4	91	94	1,289	2.7	97	6	174	5.6
9	**100**	2	66	5.7	88	74	1,083	2.8	97	1^b	23	5.5
10	**100**	0^b	9	4.4	88	96	1,436	2.9	91	35	1,012	5.4

Appendix G: Results obtained by construction heuristics

Table 12 Bin packing heuristics: average and maximum gaps to B&B solutions (TWCT objective)

w_i	$\|T\|$	FFD		BFD		Knapsack		$\text{Knapsack}^{\text{Min}}$		FFI		BFI	
		Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max
$[0, 0.3\mathcal{C}]$	10	3.09	8.13	3.09	8.13	1.42	5.52	0.10	0.95	7.05	12.28	7.05	12.28
$[0, 0.3\mathcal{C}]$	20	1.52	4.39	1.52	4.39	1.10	4.35	0.04	0.52	7.37	12.29	7.37	12.29
$[0, 0.3\mathcal{C}]$	30	1.13	2.96	1.13	2.96	0.90	3.76	0.02	0.30	7.21	10.76	7.21	10.76
$[0, 0.3\mathcal{C}]$	40	0.93	2.63	0.93	2.63	0.78	2.95	0.01	0.06	7.21	10.47	7.21	10.47
$[0, 0.5\mathcal{C}]$	10	4.33	13.12	4.33	13.12	4.91	13.06	1.10	5.50	14.29	25.18	14.29	25.18
$[0, 0.5\mathcal{C}]$	20	2.56	7.12	2.56	7.12	3.94	9.62	0.75	3.47	14.61	22.59	14.61	22.59
$[0, 0.5\mathcal{C}]$	30	1.72	4.75	1.72	4.75	3.04	7.82	0.30	1.81	13.70	19.22	13.70	19.22
$[0, 0.5\mathcal{C}]$	40	1.15	3.52	1.15	3.52	2.72	6.90	0.08	1.30	14.25	18.59	14.25	18.59
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	10	9.58	25.84	9.58	25.84	3.44	15.37	2.03	12.52	27.22	41.70	27.22	41.70
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	20	8.53	19.47	8.53	19.47	3.89	13.04	1.91	8.82	30.96	42.23	30.96	42.23
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	30	7.53	16.23	7.53	16.23	3.58	11.19	1.68	6.21	32.80	43.98	32.80	43.98
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	40	6.08	14.30	6.08	14.30	3.53	9.28	1.21	4.62	33.89	42.00	33.89	42.00
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	10	7.93	20.13	7.93	20.13	6.60	20.17	3.71	15.25	25.98	40.56	25.98	40.56
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	20	7.23	15.69	7.23	15.69	7.65	16.78	4.51	13.54	29.70	40.12	29.70	40.12
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	30	6.41	13.01	6.41	13.01	7.69	15.53	4.42	12.94	30.86	40.15	30.86	40.15
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	40	4.42	11.51	4.42	11.51	7.48	15.51	3.54	10.43	31.01	38.76	31.01	38.76
$[0, \mathcal{C}]$	10	4.48	13.23	4.48	13.23	7.30	18.41	3.10	13.01	24.63	39.32	24.63	39.32
$[0, \mathcal{C}]$	20	4.05	9.89	4.05	9.89	7.34	14.51	3.70	10.96	27.81	37.80	27.81	37.80
$[0, \mathcal{C}]$	30	3.13	7.96	3.13	7.96	6.94	13.94	3.19	8.33	28.39	38.34	28.39	38.34
$[0, \mathcal{C}]$	40	1.90	7.35	1.90	7.35	6.40	13.01	2.18	6.13	27.89	35.57	27.89	35.57

Table 13 Bin packing heuristics: average and maximum gaps to B&B solutions (makespan objective)

w_i	$\|T\|$	FFD		BFD		Knapsack		Knapsack$^{\text{Min}}$	
		Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max
$[0, 0.3\mathcal{C}]$	10	3.02	12.19	3.02	12.19	2.50	11.51	0.17	3.85
$[0, 0.3\mathcal{C}]$	20	1.74	10.07	1.74	10.07	2.09	10.24	0.08	2.59
$[0, 0.3\mathcal{C}]$	30	1.21	6.03	1.21	6.03	1.89	7.86	0.03	1.50
$[0, 0.3\mathcal{C}]$	40	1.06	5.02	1.06	5.02	1.72	8.33	0.00	0.00
$[0, 0.5\mathcal{C}]$	10	4.48	13.33	4.48	13.33	7.45	19.07	2.30	12.76
$[0, 0.5\mathcal{C}]$	20	2.93	9.27	2.93	9.27	6.74	17.59	1.89	10.05
$[0, 0.5\mathcal{C}]$	30	1.93	6.00	1.93	6.00	5.85	14.02	0.96	6.05
$[0, 0.5\mathcal{C}]$	40	0.94	4.27	0.94	4.27	5.78	14.21	0.64	5.16
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	10	3.04	13.47	3.04	13.47	4.72	20.77	2.95	18.27
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	20	2.42	9.03	2.42	9.03	6.01	20.88	3.09	15.49
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	30	2.52	7.93	2.52	7.93	6.74	19.46	3.30	13.62
$[0.3\mathcal{C}, 0.7\mathcal{C}]$	40	2.24	6.92	2.24	6.92	8.79	18.30	4.13	14.35
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	10	2.62	11.49	2.62	11.49	8.66	25.88	5.56	23.17
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	20	2.07	8.81	2.07	8.81	11.17	25.20	7.52	21.30
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	30	2.12	6.76	2.12	6.76	12.67	24.25	8.56	21.29
$[0.2\mathcal{C}, 0.8\mathcal{C}]$	40	1.59	5.39	1.59	5.39	14.51	27.52	9.41	19.28
$[0, \mathcal{C}]$	10	1.88	10.08	1.88	10.08	7.55	21.97	3.44	17.58
$[0, \mathcal{C}]$	20	1.56	6.85	1.56	6.85	9.53	22.34	5.53	16.92
$[0, \mathcal{C}]$	30	1.26	5.45	1.26	5.45	11.09	23.12	6.46	16.04
$[0, \mathcal{C}]$	40	0.55	3.17	0.55	3.17	11.23	19.81	6.04	13.68

References

Artigues, C., & Lopez, P. (2015). Energetic reasoning for energy-constrained scheduling with a continuous resource. *Journal of Scheduling, 18*(3), 225–241. https://doi.org/10.1007/s10951-014-0404-y

Artigues, C., Lopez, P., & Haït, A. (2013). The energy scheduling problem: Industrial case-study and constraint propagation techniques. *International Journal of Production Economics, 143*(1), 13–23.

Baptiste, P., Le Pape, C., & Nuijten, W. (1999). Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research, 92*, 305–333. https://doi.org/10.1023/A:1018995000688

Blazewicz, J., & Liu, Z. (1996). Scheduling multiprocessor tasks with chain constraints. *European Journal of Operational Research, 94*, 231–241.

Blazewicz, J., Drabowski, M., & Weglarz, J. (1986). Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers C, 35*(5), 389–393.

Blazewicz, J., Machowiak, M., Weglarz, J., Kovalyov, M., & Trystram, D. (2004). Scheduling malleable tasks on parallel processors to minimize the makespan. *Annals of Operations Research, 129*(1–4), 65–80. https://doi.org/10.1023/B:ANOR.0000030682.25673.c0

Blazewicz, J., Kovalyov, M., Machowiak, M., Trystram, D., & Weglarz, J. (2006). Preemptable malleable task scheduling problem. *IEEE Transactions on Computers, 55*(4), 486–490.

Blazewicz, J. W., Domschke, W., & Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research, 33*, 1–33.

Brandao, F., & Pedroso, J. P. (2016). Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research, 69*, 56–67.

Braune, R. (2019). Lower bounds for a bin packing problem with linear usage cost. *European Journal of Operational Research, 274*(1), 49–64. https://doi.org/10.1016/j.ejor.2018.10.004

Cambazard, H., Mehta, D., O'Sullivan, B., & Simonis, H. (2013). Bin packing with linear usage costs - an application to energy management in data centres. Lecture Notes in Computer Science. In C. Schulte (Ed.), *Principles and practice of constraint programming,* (Vol. 8124, pp. 47–62). Springer.

Caramia, M., & Drozdowski, M. (2006). Scheduling malleable tasks for mean flow time criterion. In: *Abstracts of the 10th international workshop on project management and scheduling,* (pp. 106–109)

Coffman, E. G., Jr., Garey, M. R., & Johnson, D. S. (1978). An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing, 7*(1), 1–17. https://doi.org/10.1137/0207001

Dell'Amico, M., Diaz, J. C. D., & Iori, M. (2012). The bin packing problem with precedence constraints. *Operations Research, 60*(6), 1491–1504.

Delorme, M., Iori, M., & Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research, 255*(1), 1–20.

Drozdowski, M. (2009). *Scheduling for parallel processing*. Computer Communications and NetworksSpringer Verlag.

Drozdowski, M., & Dell'Olmo, P. (2000). Scheduling multiprocessor tasks for mean flow time criterion. *Computers & Operations Research, 27*(6), 571–585.

Fukunaga, A. S., & Korf, R. E. (2007). Bin completion algorithms for multicontainer packing, knapsack, and covering problems. *Journal of Artificial Intelligence Research, 28*(1), 393–429.http://dl.acm.org/citation.cfm?id=1622591.1622602

Gabay, M. (2014). High-multiplicity scheduling and packing problems. PhD thesis, Université Joseph Fourier

Garey, M., Graham, R., Johnson, D., & Yao, A. C. (1976). Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A, 21*(3), 257–298. https://doi.org/10.1016/0097-3165(76)90001-7

Goemans, MX., & Rothvoß, T. (2014). Polynomiality for bin packing with a constant number of item types. In *Proceedings of the twenty-fifth annual ACM-SIAM Symposium on discrete algorithms, society for industrial and applied mathematics,* (pp. 830-839) USA, SODA '14,

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Operations Research, 5*, 187–326.

Hendel, Y., Kubiak, W., & Trystram, D. (2015). Scheduling semi-malleable jobs to minimize mean flow time. *Journal of Scheduling, 18*(4), 335–343. https://doi.org/10.1007/s10951-013-0341-1

Jansen, K. (2017). New algorithmic results for bin packing and scheduling. In D. Fotakis, A. Pagourtzis, & V. T. Paschos (Eds.), *Algorithms and Complexity* (pp. 10–15). Springer International Publishing.

Jouglet, A. (2002). The one machine total cost sequencing problem. PhD thesis, Université de Technologie de Compiègne, Compiègne, France

Jouglet, A., Baptiste, P., & Carlier, J. (2004). Branch-and-bound algorithms for total weighted tardiness. In J.T. Leung (ed) *Handbook of Scheduling,* CRC Press, chap 13

Kis, T. (2005). A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. *Mathematical Programming Series A, 103*, 515–539.

Labbé, M., Laporte, G., & Mercure, H. (1991). Capacitated vehicle routing on trees. *Operations Research, 39*(4), 616-622.

Lloyd, E. L. (1981). Concurrent task systems. *Operations Research, 29*(1), 189–201.

Martello, S., & Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. New York: John Wiley and Sons.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., & Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization, 19*, 79–102. https://doi.org/10.1016/j.disopt.2016.01.005

Pereira, J. (2016). Procedures for the bin packing problem with precedence constraints. *European Journal of Operational Research, 250*(3), 794–806. https://doi.org/10.1016/j.ejor.2015.10.048

Sadykov, R. (2012). A dominant class of schedules for malleable jobs in the problem to minimize the total weighted completion time. *Computers & Operations Research, 39*(6), 1265–1270. https://doi.org/10.1016/j.cor.2011.02.023, special Issue on Scheduling in Manufacturing Systems

Schiex, T., & Verfaillie, G. (1994). Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools, 03*(02), 187–207. https://doi.org/10.1142/S0218213094000108

Scholl, A., Klein, R., & Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research, 24*(7), 627–645. https://doi.org/10.1016/S0305-0548(96)00082-2

Scholl, A., Fliedner, M., & Boysen, N. (2010). Absalom: Balancing assembly lines with assignment restrictions. *European Journal of Operational Research, 200*(3), 688–701. https://doi.org/10.1016/j.ejor.2009.01.049

Schutt, A., Feydy, T., Stuckey, P., & Wallace, M. (2013). Solving RCPSP/max by lazy clause generation. *Journal of Scheduling, 16*(3), 273–289. https://doi.org/10.1007/s10951-012-0285-x

Shaw, P. (2004). A constraint for bin packing. In M. Wallace (Ed.), *Principles and practice of constraint programming - CP 2004* (pp. 648–662). Springer.

Wang, K., Chau, V., & Li, M. (2018). Scheduling fully parallel jobs. *Journal of Scheduling, 21*(6), 619–631. https://doi.org/10.1007/s10951-018-0563-3

Zhang, Q., Wu, W., & Li, M. (2013). Minimizing the total weighted completion time of fully parallel jobs with integer parallel units. *Theoretical Computer Science 507*, 34–40, https://doi.org/10.1016/j.tcs.2013.02.017, combinatorial Optimization and Applications